5-2020

Kernel Memory Leakage Detection for Intrusion Detection Systems (IDS)

Lee Ho
lee.ho@owasp.org

Kernel Memory Leakage Detection for Intrusion Detection Systems (IDS)

by

Lee Hong Ho

A Thesis Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

May, 2020

Abstract

Data leakage from kernel memory occurs when the memory block is not released back to the kernel after the memory block is unoccupied. The data leaked is arbitrary and confidential data such as, encryption key and password may leak out. Meltdown and Spectre are methods from side channel attacks that takes advantage of this data leakage to gain confidential data (Graz University of Technology, 2018a). This study is on how kernel memory leakage can be read as kernel memory is a protected memory area that even the root account of an operating system is unable to access (Ning, Qing, & Li, 2006). Reading kernel memory leakage is only a part of the solution to mitigate Meltdown and Spectre. To provide a solution, the leaked data from kernel memory must be of use to an Intruder Detection System (IDS) for alerts to determine if there is a possible attack on kernel memory to attain confidential data. As a result, kmemleak is used as a module created to provide a way to detect possible kernel memory leaks that is similar to a tracing garbage collector(gc) (The kernel development community, n.d.a).

Acknowledgements

I would like to thank Al-Sharabati, Abdulrahman for his technical knowledge contribution that resolved and completed the technical issues of this study much faster and accurate. It would not have occurred to me that a hard disk space can be partition or extend while the operating system is been rebuild. Many thanks for your help.

Table of Contents

List of Tables

List of Figures

Chapter I: Introduction

Introduction

Intrusion Detection Systems (IDS) started back in 1985 with lots of struggles and challenges to deliver efficient monitoring that are rule-based and managing time consuming interpretation of huge volume of log files. A constant scanning and updating the signature list is draining resources that made IDS remain in an unpopular stand until late 1980s where there is a growth for network IDS based on the need for user access and user monitoring (Threat Stack, Inc, 2015). In today's world, the challenges have grown, and we must recognize that protecting network data transmission is a part of the solution. In addition to protecting network data transmission, there is a need to protect computer memory from attacks. Memory attack can be performed on a computer or any other electronic device such as a mobile phone and Internet of Things (IoT). For example, Meltdown uses side-channel techniques to attack leaking arbitrary data from kernel memory (Schwarz, Canella, Giner, & Gruss, Store-to-Leak Forwarding: LeakingDataonMeltdown-resistantCPUs, 2019.a). A patch known as CVE-2017-5754 is created to prevent further attacks from this Meltdown variant (NIST, 2019). However, the patch provided resolves a specific vulnerability in side-channel attack. However, the original Meltdown variant created other variants of Meltdown as the kernel memory is still vulnerable (Schwartz, 2018). Thus, there is a need to perform software fixes to ensure that the kernel and user space are isolated properly. Another side-channel attack on computer memory is known as Spectre with two patches known as CVE-2017-5753 and CVE-2017-5715 (Technology, 2018). As side-channel attack grows into a major threat for secure embedded systems (The MITRE Corporation , 2014), additional solutions to patch management are needed. In 2014, a research for a fast, inexpensive, and automated technique to test software for side-channel attacks vulnerability was developed. This side-channel leakage evaluator and analysis kit is called SLEAK (Walters, Hagen, & Kedaigle, 2014). In addition to software testing, a Host-based IDS is useful because the Host-based IDS monitors the system after the system is live and running. Therefore, Host-based IDS is an additional security to SLEAK to identify when a kernel memory leakage has occurred and provide an alert for further investigation to determine if this kernel memory leakage has exposed any confidential data.

As kernel memory attack increases, patch management of threats are only a part of the solution. As security solutions are a layered solution, there is an importance to identify more solutions that can fit into different layer of the security solution. Therefore, the scope of this study is a proof-of-concept to prove if kernel memory leakage can be read to create an alert warning system for an IDS.

Problem Statement

Data leakage in kernel memory resulting in exposed confidential data such as, encryption key or password of a user from side-channel attack.

Nature and Significance of the Problem

Side channel attack such as Meltdown and Spectre are known to exploit microarchitectural changes the CPU makes during transient out-of-order execution resulting in leaking arbitrary data from memory. As a workaround, software mitigations are disabled on recent processors. However, Meltdown-like attacks are still possible on recent CPUs that are not vulnerable to the original Meltdown attack. As a solution, software fixes are necessary to ensure proper isolation between the kernel and user space (Schwarz, Canella, Giner, & Gruss, Store-to-LeakForwarding: LeakingDataonMeltdown-resistantCPUs, 2019.b).

This study is useful as the result of this study provides a basis to design an early warning system for kernel memory attacks. For example, a Host-based IDS can act as an early warning system with Indicators of Attack (IoA) as filtering rules for anomalies.

Objective of the Study

The objective of this study is to perform a qualitative study approach to determine if kernel memory leakage can be read in log files and if security tools, such as a host-based IDS can create alerts based on the log files.

Study Questions/Hypotheses

The first research question is to determine how can the kernel memory leakage be read? Then if the kernel memory leakage can be read, will there be any log files created and where are the files stored? The final question is to determine if the log files can be read by an IDS? Thus, this study creates a solution on how to detect kernel memory leakage for the use of an IDS.

Limitations of the Study

The first limitation is kmemleak by design is limited to the following processors x86, arm, powerpc, sparc, sh, microblaze, ppc, mips, s390 and tile (The kernel development community, n.d.b). The next limitation is any major updates to the Linux kernel version 4.15.0 or the firmware will make changes that may not be compatible with this study.

Definition of Terms

Intrusion Detection Systems (IDS): A device or software application used to monitor malicious activity and anomalies in computer activities.

Institute of Electrical and electronics Engineers (IEEE): A professional association for electronic engineering and electrical engineering.

Indicators of Attack (IoA): A unique construction of unknown attributes, IoCs, and contextual information, including organizational intelligence and risk, into a dynamic, situational picture that guides response (McAfee, 2014).

Indicators of Compromise (IoC): Forensic evidence of potential intrusions on a host system or network (Trend Micro, 2019).

kmemleak: The Linux module that scans the kernel memory for memory leakage.

LSM: Linux Security Module is a list of mechanisms to reduce the kernel attack surface and improve security (Smalley, Fraser, & Vance, n.d.).

CONFIG_DEBUG_KMEMLEAK: This is the option that is enabled in kernel hacking in order to scan the kernel memory leakage.

Garbage Collector (GC): A program written to automatically free up unused memory block in user space.

Summary

Network IDS became popular because there is a growth in networks in late 1980s that develop a need for user access and user monitoring. Now there is a new question to ask, that is will the increasing threat of side-channel attacks on kernel memory be enough to create a demand to develop a monitoring tool for the kernel memory to be re-compile into a Host-based IDS? To help understand the increasing threats and demand better, let’s review the background and literature related to the problem.

Chapter II: Background and Review of Literature

Introduction

This chapter introduces what is kernel memory leakage, why kernel memory leakage happens, the existing threats to kernel memory leakage and the components needed to provide a solution for the problem statement.

Background Related to the Problem

Kernel memory leaks happen when memory is no longer needed and is not release back to the kernel. When memory leaks in an application at the user level, the garbage collector (GC) will release the memory back to the system. However, GC function is at the user level and not at the kernel level. That is why all leaked kernel memory is not available to the system until the next reboot. The cause of the memory leakage can be of several reasons and one of these reasons is that when the LKM is unloaded, the changes made were not undone and the kernel memory will remain locked by the unloaded LKM. Furthermore, tracking down memory leakage is difficult and as kernel developers depend more on automated tools to find bugs, the need to create a kernel memory leak detector becomes more obvious.

Literature Related to the Problem

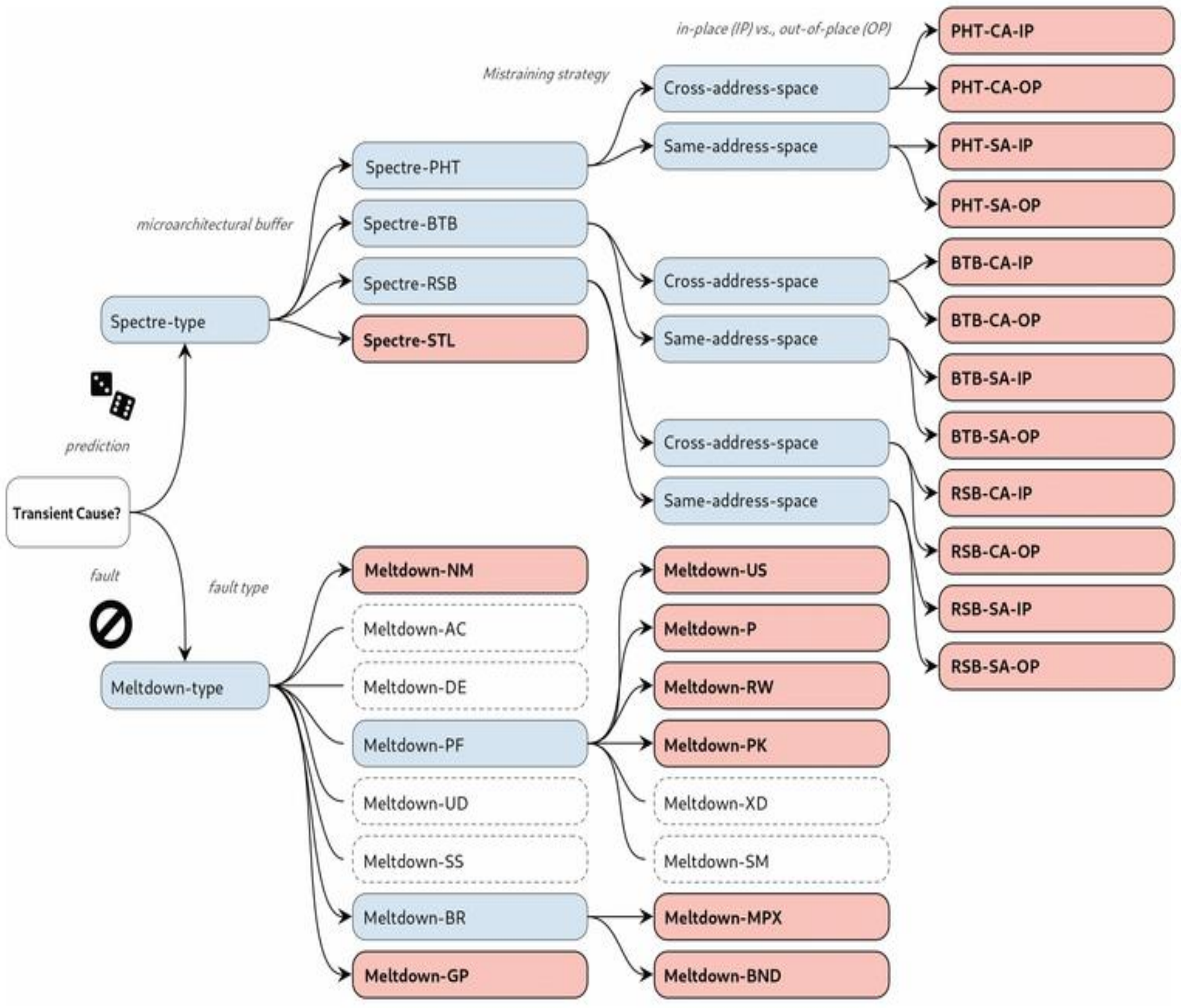
Meltdown and Spectre are side-channel attacks techniques that utilizes kernel memory leakage to gain confidential data. The systems that are affected by Meltdown are computers, laptops and clouds with out-of-order execution implemented. In general, this means all Intel processors created since 1995 are affected. That is 25 years of processors that were created are vulnerable to Meltdown (Graz University of Technology, 2018b).

The variants of both side-channel attack, Meltdown and Spectre are increasing and as of May 15, 2019, there are 13 Spectre variants and 14 Meltdown variants. In addition to the number of variants, the number of affected devices are expected to increase as Internet of Things (IoT) (Sanders, 2019) and both Android and Apple mobile phones (Abrams, 2018) are affected. To understand these variants better, a brief description of these common variants will provide a general idea of how both Spectre and Meltdown operates (Kerner, 2018).

- The first variant of the three original variants is Spectre variant 1 (Bounds Check Bypass/CVE-2017-5753). This variant allows the attacker to abuse an operating system kernel to read the memory of another system process.
- The next variant is Spectre variant 2 (Branch Target Injection/CVE-2017-5715). This variant abuses the CPU branch predictor function to allow an attacker to read data from an operating system kernel.
- The last original variant is Meltdown (Rogue Data Cache Load/CVE-2017-5754). Meltdown abuses memory page tables to read operating system kernel data from user space.

- The variant, Meltdown (Rogue System Register Read/CVE-2018-3640) allows an attacker with local access to speculatively read system parameters via side channel analysis and obtain confidential information.
- The fourth variant is Speculative Store Bypass/CVE-2018-3639. This variant allows the attacker to read older memory values in a CPU’s stack or other memory location. Though the implementation is complex, less privileged code could read arbitrary privileged data and run older commands resulting in using cache allocations to exfiltrate data by standard side channel methods.
- The fifth variant is Lazy Floating Point State Restore/CVE-2018-3665. Intel advisory warn, “System software may utilize the Lazy FP state restore technique to delay the restoring of state until an instruction operating on that stat is actually executed by the new process.”.
- The last common variant is L1TF – L1 Terminal Fault “Foreshadow”/CVE-2018-3615, CVE-2018-3620, CVE-2018-3646. Foreshadow is a collection of three CVE vulnerabilities that enables an attacker to extract privileged information from vulnerable CPUs, hypervisors and Intel Software Guard Extension (SGX) secure enclave technologies.

For a description of all the variants, refer to the Systematic Evaluation paper (Canella, et al., 2019). Below, in figure 1 on page 16 is a diagram of Spectre and Meltdown categorization.



Classification tree of Spectre and Meltdown variants, with demonstrated attacks (red, bold), and negative results (white).
Graph Data: Canella et al. Edited Image: James Sanders/TechRepublic

Figure 1: Spectre and Meltdown classification

A paper was published by IEEE in 2010 about the importance of monitoring for intrusion, malware infection and information leakage with the objective to explain the importance of protecting the monitoring module and results for security inspection. This applies to this study as the monitoring module will be like the Host-based IDS and the results are syslog and kmemleak output. In this article, Linux Security Module (LSM) is a set of hook functions implemented to monitor and control the system calls. The author proposed to secure the monitoring scheme that is implemented with LSM-based function and protected by kernel protection techniques. The integrity of the results is then protected by using a Mandatory Access Control (MAC) of Linux kernel and a mechanism of the trusted process invocation (Isohara, Takemori, Miyake, Qu, & Perrig, 2010).

Literature Related to the Methodology

Loadable Kernel Module (LKM) is a mechanism that adds and removes programming code from the kernel at runtime. This feature allows LKM to be loaded into an electronic device without restarting the device. Therefore, LKM is convenient if the user does not need the features of the LKM to be installed into the kernel and yet need the features of the LKM (Molloy, 2015). For the purpose of this study, LKM serves the purpose of accessing the kernel memory. However, accessing the kernel memory is a kernel mode that consist of rights that are permitted by the kernel. In general, the protected area of a kernel memory is not accessible to all users, even the root user. Reason is that the root user is from the user level of the operating system and not from the kernel level. In addition to the inaccessibility of the kernel, the data collection of this study needs to run the LKM continuously for a few days to collect data of memory leakage for analysis. This brings up the next available option of kmemleak.

In 2006, Catalin Marinas developed a debug similar to GC tri-color concept of scan-and-mark with the objective to detect kernel memory leakage (Corbet, 2006). This debug program is now known as kmemleak. The difference between GC and kmemleak is that the orphan objects found are not freed but reported via kmemleak at /sys/kernel/debug/kmemleak. The data leaked from kernel memory is random. However, with side-channel attack techniques, confidential data can be leaked as explained by Systematic Evaluation paper. For example, in brief, a kmemleak-test module that deliberately leaks memory (The kernel development community, n.d.a) can be installed into the attacker's code and executed after the attacker perform social engineering to a user to make the user login to their computer system. Kmemleak is the preferred method tool for this study as kmemleak will be re-compile into the kernel.

Kmemleak provides a report of the actual data that is leaked. This report is done manually when there is a need to view what type of data is been leaked out. In order to know when to view kmemleak, syslog is used. Syslog is a log file that stores event messages from network devices. Both network and Host-based IDS collects data with syslog protocol and the syslog format is a general format used by other monitoring tool including Splunk as a Security Information and Event Management (SIEM) tool. Therefore, syslog will report any new suspected memory leaks that can be used by the Host-IDS to create an alert.

An open source Host-based IDS named OSSEC is selected to explain the study of this paper. OSSEC is capable of monitoring and analyzing data from multiple log data points in real-time. This feature will allow OSSEC to monitor and analyze syslog in real-time and produce the alert immediately. Other OSSEC capabilities are detecting rootkit and malware, integration with 3rd party software and firewalls, application and system level auditing, file integrity monitoring and collects system information (OSSEC PROJECT TEAM, 2020).

Summary

Reviewing the reasons for kernel memory leakage and the threats that are a risk to exposing confidential data through side-channel attacks lead to the literature in explaining the background of how to identify and create the alerts for a Host-based IDS. The next step is to explain the methodology on how to detect kernel memory leakage for IDS. There are two options in this methodology, the first option is to detect kernel memory leakage with a Loadable Kernel Module (LKM). The other option is to re-compile the Linux kernel with kmemleak module. There are advantages and disadvantages in both these techniques, that are explained in more detail in the next chapter of methodology.

Chapter III: Methodology

Introduction

This chapter explains about what options are available to read the kernel memory leakage, how the kernel memory leakage is read, and the reasons that help determine which is the preferred method to use to complete this study. These explanations are discussed in figure 2, that is the methodology framework on page 23. In addition to the study design that answers the problem statement, the recommended solution that includes the host-based IDS and creation of bash scripts are explained.

Design of the Study

A qualitative approach is taken to conduct this study because as of Mar 2020, there are no other published paper or conference that research into how to monitor and detect kernel memory leakage for a Host-based IDS. To design the study framework, first the problem statement of "confidential data is leaked from kernel memory" must be understood. Confidential data is the result of an action that is leaked from the subject kernel memory. Therefore, the focus is on how to access the kernel memory? In order to determine if the kernel memory leakage can be accessed and read, it is important to understand that the kernel memory is in a protected memory area that is not accessible even by the root account of the operating system (Ning, Qing, & Li, 2006) and is located in between the hardware and the operating system. Even though the kernel is secured by design, the kernel consists of functions that links both hardware and the operating system. Refer to figure 24 in the Appendix A on page 55 to understand that system calls are used by the kernel memory to communicate with the operating system that is used in device drivers in the kernel memory to communicate between the operating system and hardware control. Then the kernel memory will use interrupt numbers with the hardware control to communicate with the hardware. As device driver is in the kernel memory and communicates with the operating system, this brings up the idea of an object file that contains code to extend the running kernel. This object file is called Loadable Kernel Module (LKM) (Clinton, 2018) shown in figure 2 on page 23, under the Option 1 heading. The advantage of LKM is that the module is loaded and unloaded as needed without the need to reboot the computer (ArchWiki, 2020). The kernel size will remain the same as codes are not written into the kernel. However, LKM has its disadvantages, that is the module management of LKM consumes un-pageable kernel memory. This means that when the LKM is unloaded, the changes that were made must be undone. Otherwise the memory will not be released back to the kernel causing a possible kernel memory leakage. In addition, loading numerous modules will consume more memory compare to re-compiling the modules into the kernel image itself (Silberschatz, Galvin, & Gagne, 2012). The other drawback of LKM is that the LKM can be used to modify the running kernel by attackers and prevent detection of the attacker’s processes and files. This is made popular by LKM rootkits where operating system modules do not allow privilege elevation that is required to load LKM. Therefore, LKM became a targeted attack vector (Phrack Inc). For example, an attacker performs code injection and remains stealth by renaming the initialization module and insert the attacker’s module that calls for the initialization module. This ensures that the LKM behavior remains the same with additional functions from the attacker (Phrack

Magazine, 2003). The malicious intent used of LKM as a kernel mode rootkit is further emphasized on by MITRE. MITRE is a globally-accessible knowledge base of adversary tactics and techniques based on real world observations of specific threat models and methodologies in the private sector, government, cybersecurity product and service community (MITRE Corporation, 2015 - 2020). Therefore, the usage of LKM is dependent on the frequency of use and if there are any pre-linked device driver with the kernel (Silberschatz, Galvin, & Gagne, 2012). For the purpose of this study, the second option to re-compile the kernel is selected as for the data collection stage will require the system to be run continuously for a few days. LKM has a higher possibility to stop responding compare to a re-compiled kernel.

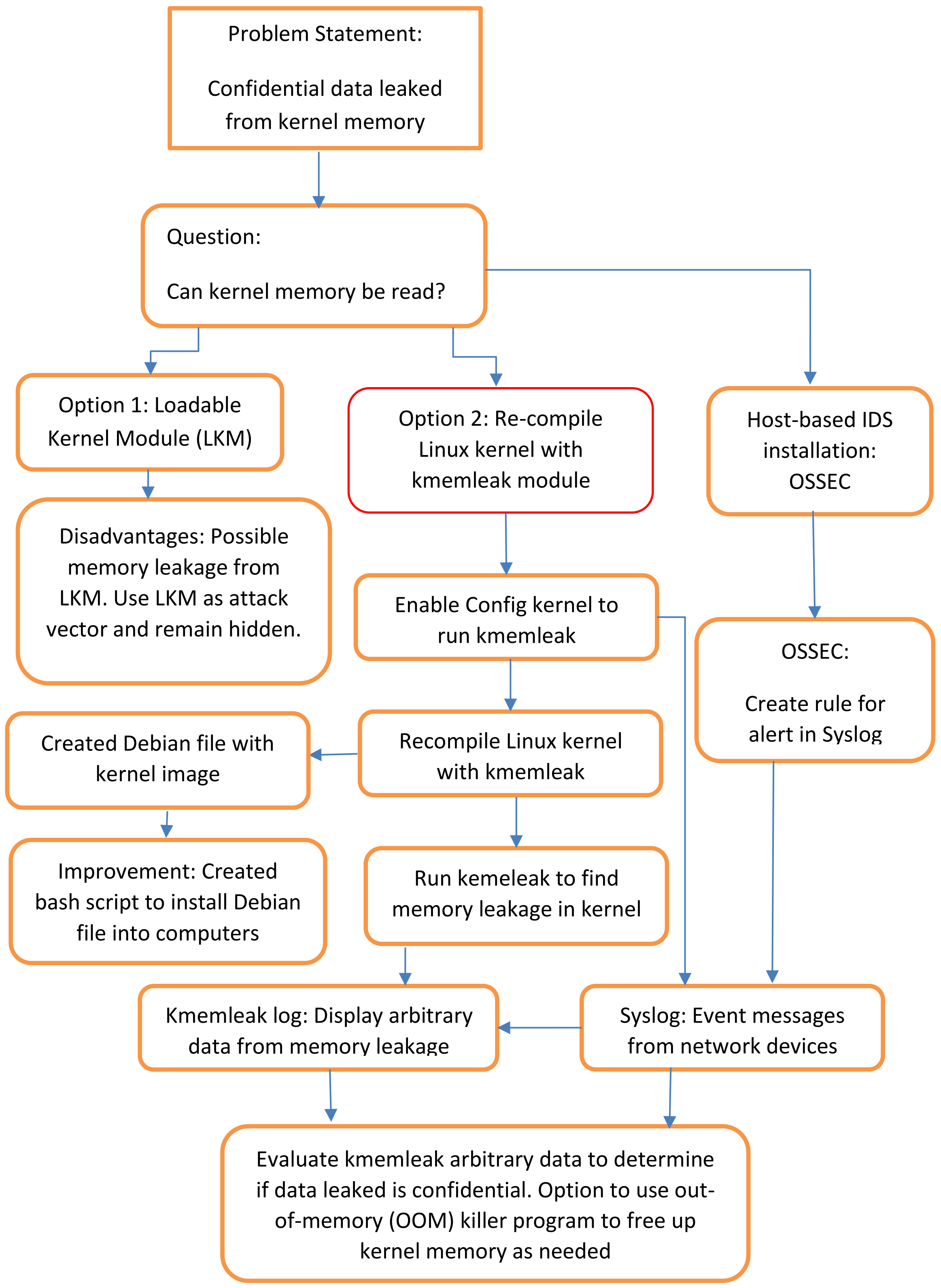

Problem Statement:
Confidential data leaked from kernel memory
Question:
Can kernel memory be read?
Option 1: Loadable Kernel Module (LKM)
Option 2: Re-compile Linux kernel with kmemleak module
Host-based IDS installation: OSSEC
Disadvantages: Possible memory leakage from LKM. Use LKM as attack vector and remain hidden.
Enable Config kernel to run kmemleak
OSSEC:
Create rule for alert in Syslog
Created Debian file with kernel image
Recompile Linux kernel with kmemleak
Improvement: Created bash script to install Debian file into computers
Run kemeleak to find memory leakage in kernel
Kmemleak log: Display arbitrary data from memory leakage
Syslog: Event messages from network devices
Evaluate kmemleak arbitrary data to determine if data leaked is confidential. Option to use out-of-memory (OOM) killer program to free up kernel memory as needed

Figure 2: Methodology Framework

The other option is to build and re-compile the Linux kernel with a module that can identify and read kernel memory leakage. This module is known as kmemleak (The kernel development community, 2017). A pre-configuration to the “Config” kernel is needed to enable the kmemleak module in the kernel configuration. This is shown in figure 2 on page 23, under the heading “Option 2: Re-compile Linux operating system with kmemleak module”. For a step by step screenshot on what to select, please refer to figure 11, 12 and 13 on page 48 and 49 that shows the selection of Kernel hacking to Memory Debugging to enable Kernel Memory Leak detector. The next step to re-compile Linux kernel from source file with kmemleak is shown in figure 14 on page 49. As the figure shows the source code files with the extension .o are been convert to kernel modules with .ko extension. At the end of the re-compilation that took a day to re-compile, a Debian file with the kernel image is created. Now with a Debian file, the kmemleak can be installed to a computer within an hour without the need to re-compile the Linux kernel (The kernel development community, n.d.). For improvement, a bash script file is created to make the installation of the Debian file to a new computer easier as shown figure 15 and figure 16 on page 50. Figure 15 shows the bash script named runme2.sh and figure 16 shows the contents in the script file. As the kmemleak code is re-compile into the kernel, the kernel size grew pass 20 GB of hard disk space and the re-compilation stalled as shown in figure 17 on page 51. Fortunately, this is a virtual machine where the hard disk space can be re-partitioned using gparted to 32 GB while the re-compiling is running. Therefore, planning the kernel hard disk space is important as re-compiling the kernel will increase the hard disk space. The following figures 18 to figures 21 on page 51 to 53 are commands needed to complete the re-compiling. These commands are included in the bash script written to install the Debian file that is shown in figure 16 on page 50. After re-compiling the kernel, kmemleak module is executed with the command “echo scan > /sys/kernel/debug/kmemleak” as shown in figure 22 on page 53 to view the actual data that is leaked in memory. The output of the actual data that is leaked from kernel memory is shown in figure 23 on page 54. In addition to the log file that shows the actual data leakage, the other log file format is from syslog. Syslog is a log file that stores event messages from network devices. Figure 3, 4 and 5 on page 29 and 30 under the data presentation heading shows a sample of the syslog. For the purpose of this study, these syslog files consist of the event messages needed by the Host-based IDS to create alerts.

In figure 2 on page 23, OSSEC is shown as the Host-based IDS that is for collecting and analyzing the data log in plain text or syslog format (OSSEC Project Team, 2010-2020). As OSSEC reads syslog format, the syslog files collected from network devices can be collected and analyzed at a central location using SIEM such as Splunk (Lhotsky, 2013). The reason for selecting a Host-based IDS (OSSEC) than a network based IDS is that OSSEC can be installed on the same host with kmemleak. For OSSEC to read the syslog files, an entry must be added into ossec.conf file located at /var/ossec/etc. For example, the script in ossec.conf will look like:

```
<localfile>
  <log_format>syslog</log_format>
  <location>/path_to_log_file</location>
```

```
</localfile>
```

The ossec.conf will allow OSSEC to read the syslog that identifies the new suspected memory leak. The next step is to write a custom decoder to generate the output from syslog showing the new suspected memory leak. The decoder is added into the local_decoder.xml file located at /var/ossec/etc. However, the location of this file can be changed by the administrator. Recommendation is to protect this .xml file from tampering by using OSSEC file integrity check. To generate an alert with an output in OSSEC, the following code is added into local_decoder.xml file.

```
<decoder name =”OSSEC-kernel-leak”>
   <program_name>kmemleak</program_name>
</decoder>
```

As for the purpose of this study, the syslog generated from the kernel memory leakage is been read into a Host-based IDS to alert the user of a kernel memory leakage.

Data Collection

The first step of data collection process is to gather event messages from syslog. This serves the purpose of an alert message for kernel memory leakage. Ensure that the user account has root access and navigate to the syslog file location at /var/log/sys/ folder. Then using a text editor, open the syslog file and perform a search for the keyword “/sys/kernel/debug/kmemleak”. This keyword is the location of the kmemleak file that is used to scan for kernel memory leakage. This entry will indicate how many new kernel leaks were found. Once an entry is found, this indicates that there is a kernel memory leakage. Search for all the entries with this keyword in syslog. Next, to confirm that the kernel memory is leaking, trigger an immediate memory scan using kmemleak. This is performed with an echo scan command that is executed as “echo scan > /sys/kernel/debug/kmemleak”. For better analysis, a redirector command is used to send kmemleak output result to a text file. After the memory scan is triggered, the following command “cat /sys/kernel/debug/kmemleak” is executed to view the actual data leaked from kernel memory. The output of the actual data leaked is kept in kmemleak file until the data is cleared with the following command “echo clear > /sys/kernel/debug/kmemleak”.

Tools and Techniques

The tools used to collect the data for this study are syslog and the Kmemleak module. Syslog is the log file format that is used by most network devices including IDS and firewall. Whereas, kmemleak module is a tool that is designed to detect kernel memory leakage that uses the same concept from GC. Both tools are log files that is viewed using notepad. For analysis, the data were categorized into number of hours monitored, number of leaks and number of times scanned based on date and time.

Summary

Now that the methodology of solving the problem statement is explained, the next step is to determine if the data from the kernel memory leakage can be read. This is shown in the next chapter that the kernel memory leakage can be read and log into a text file for the actual data leaked and syslog for the event message stating that a kernel memory leakage has occurred. After presenting the data, the data will be analyzed to determine the amount of data that is leaked.

Chapter IV: Data Presentation and Analysis

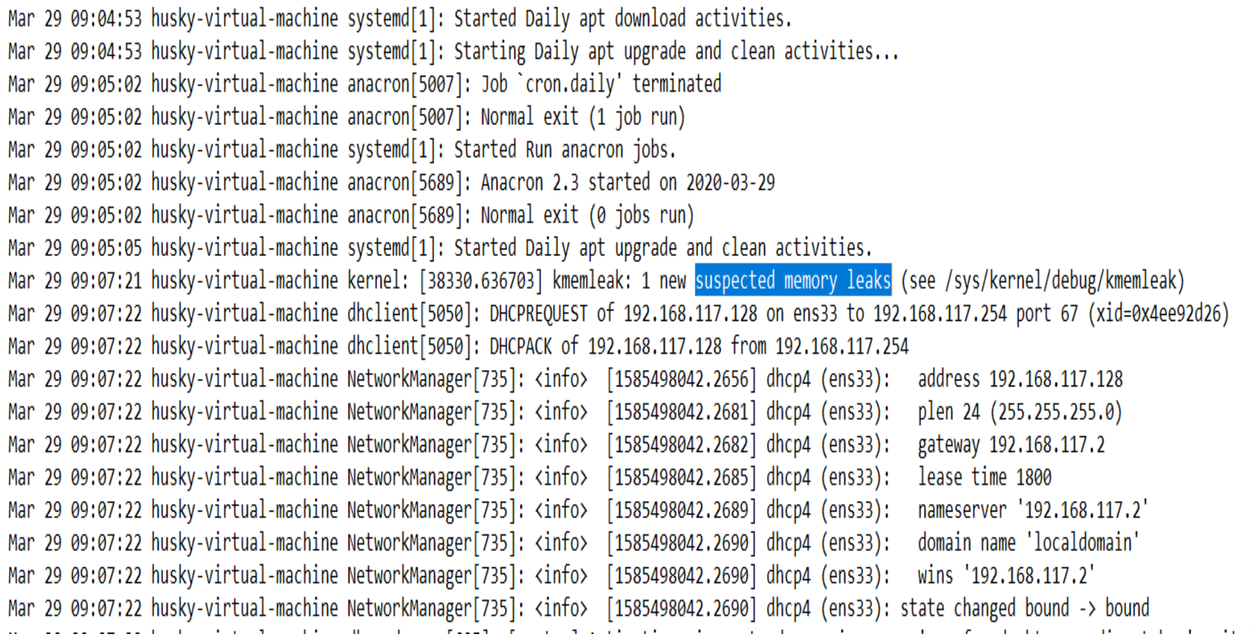
Introduction

This chapter covers the data collected from 2 log formats. The first log format is syslog that receives event messages from network devices as data. The second log format is from kmemleak that records the actual data leaked from kernel memory. The actual data leaked is arbitrary and if there are any confidential data leaked, kmemleak will display the data in this log.

Data Presentation

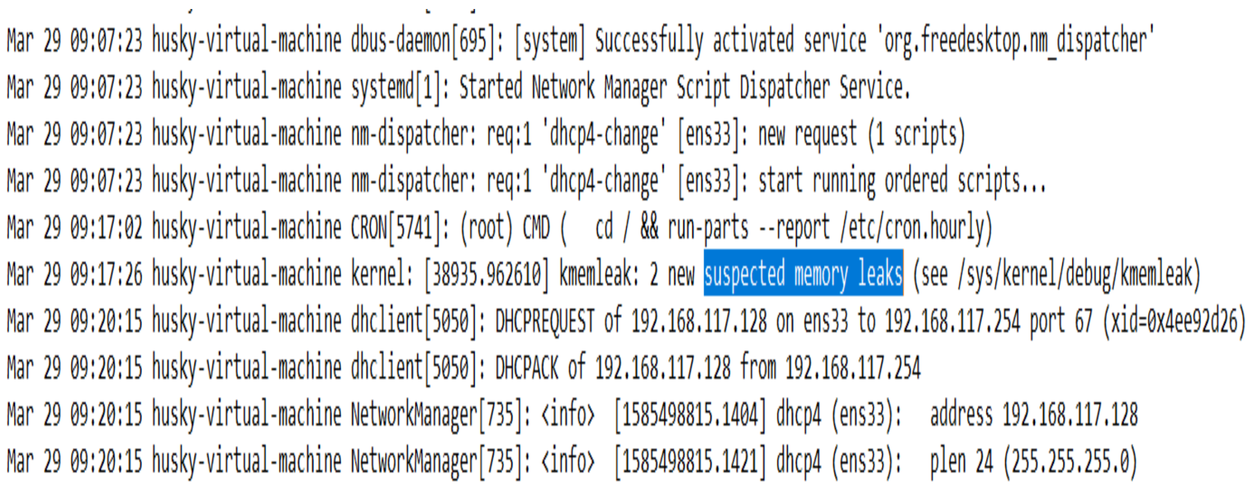
The data collected are in the form of syslog that contains event messages from network devices. As syslog only shows the event messages, the other form of data collection is from kmemleak that shows the actual arbitrary data leakage to determine if any confidential data is exposed as shown in Appendix A, figure 23 on page 54.

Syslog files collected in the first 23 hours. Data are collected into two formats. The first format is generated automatically by network devices and stored in syslog. Network devices just send event messages to the syslog. This log file will store the information of a possible kernel memory leakage as shown below in the figure 3, 4 and 5 on page 29 and 30.



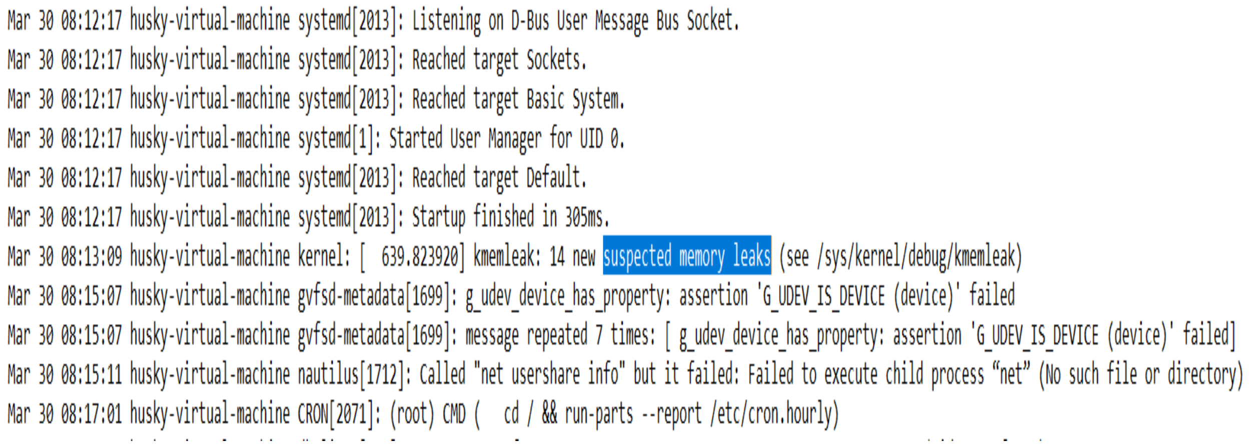
```
Mar 29 09:04:53 husky-virtual-machine systemd[1]: Started Daily apt download activities.
Mar 29 09:04:53 husky-virtual-machine systemd[1]: Starting Daily apt upgrade and clean activities...
Mar 29 09:05:02 husky-virtual-machine anacron[5007]: Job `cron.daily' terminated
Mar 29 09:05:02 husky-virtual-machine anacron[5007]: Normal exit (1 job run)
Mar 29 09:05:02 husky-virtual-machine systemd[1]: Started Run anacron jobs.
Mar 29 09:05:02 husky-virtual-machine anacron[5689]: Anacron 2.3 started on 2020-03-29
Mar 29 09:05:02 husky-virtual-machine anacron[5689]: Normal exit (0 jobs run)
Mar 29 09:05:05 husky-virtual-machine systemd[1]: Started Daily apt upgrade and clean activities.
Mar 29 09:07:21 husky-virtual-machine kernel: [38330.636703] kmemleak: 1 new suspected memory leaks (see /sys/kernel/debug/kmemleak)
Mar 29 09:07:22 husky-virtual-machine dhclient[5050]: DHCPREQUEST of 192.168.117.128 on ens33 to 192.168.117.254 port 67 (xid=0x4ee92d26)
Mar 29 09:07:22 husky-virtual-machine dhclient[5050]: DHCPACK of 192.168.117.128 from 192.168.117.254
Mar 29 09:07:22 husky-virtual-machine NetworkManager[735]: <info>  [1585498042.2656] dhcp4 (ens33):   address 192.168.117.128
Mar 29 09:07:22 husky-virtual-machine NetworkManager[735]: <info>  [1585498042.2681] dhcp4 (ens33):   plen 24 (255.255.255.0)
Mar 29 09:07:22 husky-virtual-machine NetworkManager[735]: <info>  [1585498042.2682] dhcp4 (ens33):   gateway 192.168.117.2
Mar 29 09:07:22 husky-virtual-machine NetworkManager[735]: <info>  [1585498042.2685] dhcp4 (ens33):   lease time 1800
Mar 29 09:07:22 husky-virtual-machine NetworkManager[735]: <info>  [1585498042.2689] dhcp4 (ens33):   nameserver '192.168.117.2'
Mar 29 09:07:22 husky-virtual-machine NetworkManager[735]: <info>  [1585498042.2690] dhcp4 (ens33):   domain name 'localdomain'
Mar 29 09:07:22 husky-virtual-machine NetworkManager[735]: <info>  [1585498042.2690] dhcp4 (ens33):   wins '192.168.117.2'
Mar 29 09:07:22 husky-virtual-machine NetworkManager[735]: <info>  [1585498042.2690] dhcp4 (ens33): state changed bound -> bound
```

Figure 3: First suspected memory leakage



```
Mar 29 09:07:23 husky-virtual-machine dbus-daemon[695]: [system] Successfully activated service 'org.freedesktop.nm_dispatcher'
Mar 29 09:07:23 husky-virtual-machine systemd[1]: Started Network Manager Script Dispatcher Service.
Mar 29 09:07:23 husky-virtual-machine nm-dispatcher: req:1 'dhcp4-change' [ens33]: new request (1 scripts)
Mar 29 09:07:23 husky-virtual-machine nm-dispatcher: req:1 'dhcp4-change' [ens33]: start running ordered scripts...
Mar 29 09:17:02 husky-virtual-machine CRON[5741]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Mar 29 09:17:26 husky-virtual-machine kernel: [38935.962610] kmemleak: 2 new suspected memory leaks (see /sys/kernel/debug/kmemleak)
Mar 29 09:20:15 husky-virtual-machine dhclient[5050]: DHCPREQUEST of 192.168.117.128 on ens33 to 192.168.117.254 port 67 (xid=0x4ee92d26)
Mar 29 09:20:15 husky-virtual-machine dhclient[5050]: DHCPACK of 192.168.117.128 from 192.168.117.254
Mar 29 09:20:15 husky-virtual-machine NetworkManager[735]: <info>  [1585498815.1404] dhcp4 (ens33):   address 192.168.117.128
Mar 29 09:20:15 husky-virtual-machine NetworkManager[735]: <info>  [1585498815.1421] dhcp4 (ens33):   plen 24 (255.255.255.0)
```

Figure 4: Second suspected memory leakage detected 10 minutes later



```
Mar 30 08:12:17 husky-virtual-machine systemd[2013]: Listening on D-Bus User Message Bus Socket.
Mar 30 08:12:17 husky-virtual-machine systemd[2013]: Reached target Sockets.
Mar 30 08:12:17 husky-virtual-machine systemd[2013]: Reached target Basic System.
Mar 30 08:12:17 husky-virtual-machine systemd[1]: Started User Manager for UID 0.
Mar 30 08:12:17 husky-virtual-machine systemd[2013]: Reached target Default.
Mar 30 08:12:17 husky-virtual-machine systemd[2013]: Startup finished in 305ms.
Mar 30 08:13:09 husky-virtual-machine kernel: [  639.823920] kmemleak: 14 new suspected memory leaks (see /sys/kernel/debug/kmemleak)
Mar 30 08:15:07 husky-virtual-machine gvfsd-metadata[1699]: g_udev_device_has_property: assertion 'G_UDEV_IS_DEVICE (device)' failed
Mar 30 08:15:07 husky-virtual-machine gvfsd-metadata[1699]: message repeated 7 times: [ g_udev_device_has_property: assertion 'G_UDEV_IS_DEVICE (device)' failed]
Mar 30 08:15:11 husky-virtual-machine nautilus[1712]: Called "net usershare info" but it failed: Failed to execute child process "net" (No such file or directory)
Mar 30 08:17:01 husky-virtual-machine CRON[2071]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
```

Figure 5: Twenty-three hours later, 12 new suspected memory leaks (total 14)

Kmemleak files showing actual data leak collected in the first 23 hours. The second method of data collection is the actual arbitrary data leaked from kernel memory. After reviewing the syslog, this data collection method is executed by issuing the following command “echo scan > /sys/kernel/debug/kmemleak” to view the kernel leaked data. The first record of the 14 new suspected memory leaks is shown in table 1 below on page 31. The remaining 13 new suspected memory leaks are found in the Appendix B, table 3 on page 56.

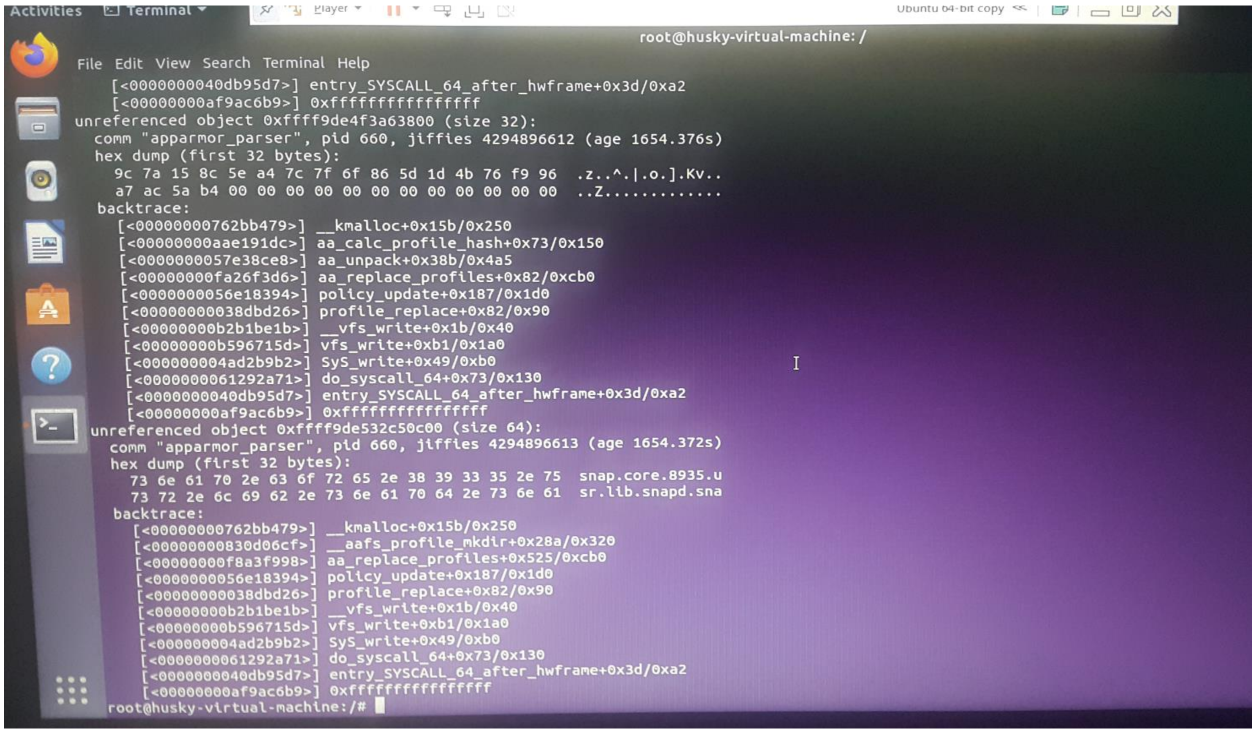
Table 1: The record of the first suspected memory leak

```
unreferenced object 0xffff942aba315c18 (size 72):
  comm "swapper/0", pid 1, jiffies 4294892395 (age 1173.260s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 0e 02 01 00 02 00 00 01  ................
    9b 57 02 40 54 b6 ff ff 0e 00 00 00 00 00 00 00  .W.@T...........
  backtrace:
    [<000000002517a863>] kmem_cache_alloc+0xf8/0x1e0
    [<000000002b330b88>] acpi_ut_allocate_object_desc_dbg+0x62/0x10c
    [<00000000f5fb5003>] acpi_ut_create_internal_object_dbg+0x53/0x117
    [<0000000025f2a706>] acpi_ds_build_internal_object+0xe5/0x1cc
    [<0000000019bb4fb0>] acpi_ds_build_internal_package_obj+0x1ea/0x331
    [<00000000f66c35a7>] acpi_ds_eval_data_object_operands+0x17d/0x219
    [<000000005c7c7ddb>] acpi_ds_exec_end_op+0x4b5/0x764
    [<000000007b93b1b9>] acpi_ps_parse_loop+0xa38/0xadc
    [<000000009fbe3906>] acpi_ps_parse_aml+0x1ac/0x4bd
    [<000000009f450676>] acpi_ds_execute_arguments+0x18b/0x1d2
    [<00000000033ee6ee>] acpi_ds_get_package_arguments+0xfd/0x12c
    [<00000000badb67db>] acpi_ns_init_one_object+0xeb/0x155
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000edaf2427>] acpi_walk_namespace+0xf9/0x144
```

```
[<00000000c7ffe1b1>] acpi_ns_initialize_objects+0x108/0x1f5

[<00000000fd360aaf>] acpi_initialize_objects+0x4b/0xd5
```

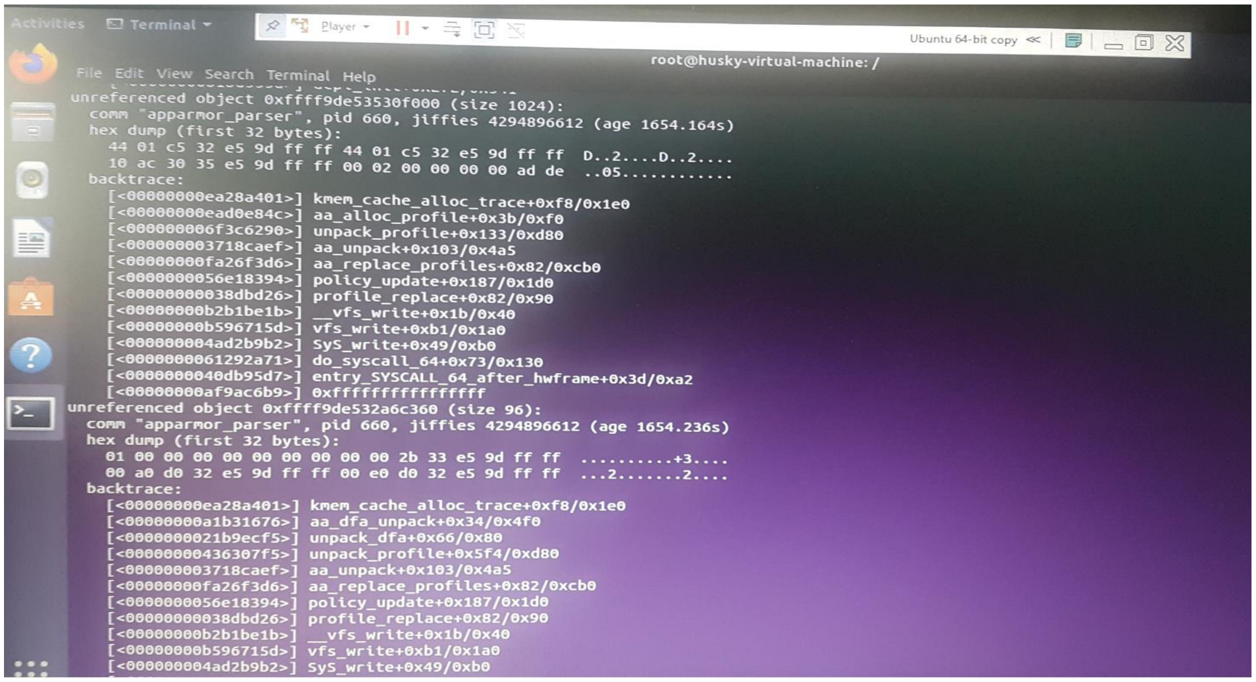
Syslog files collected in the next 73 hours. For the next 73 hours, the syslog file did not show any entries from kmemleak. Reason is that the system stopped responding and network devices could not deliver the event messages to syslog. However, the exact time when the system stopped responding is unknown as the syslog showed only the time and date that is after the 73 hours.

Kmemleak files showing actual data leak collected in the next 73 hours. There is a total of 16 leaks during these 73 hours as shown by kmemleak. 6 leaks showed arbitrary data with 10 leaks showing empty data. The 6 leaks are displayed in the following figure 6, 7 and 8 on page 33 and 34. Even though the data leaked from the kernel memory is arbitrary, the second memory leaked data in figure 6 on page 33, coincidently leaked data that looks sensible. Therefore, this does show that even randomly data leaked out can be of use. The remaining kernel memory leak that did not show any data are placed in Appendix B as additional information on page 69, 70 and 71 as figure 25, 26, 27, 28 and 29.


Activities
Terminal
Player
Ubuntu 64-bit copy
root@husky-virtual-machine: /
File Edit View Search Terminal Help

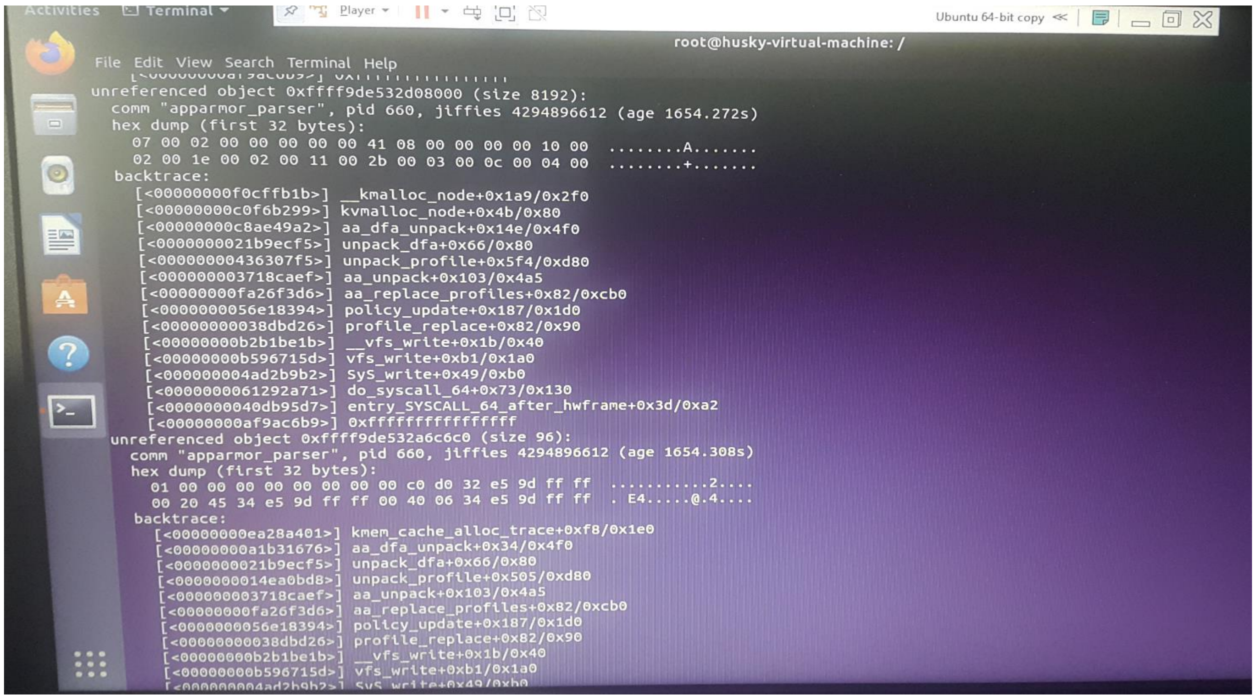
```
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de4f3a63800 (size 32):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.376s)
  hex dump (first 32 bytes):
    9c 7a 15 8c 5e a4 7c 7f 6f 86 5d 1d 4b 76 f9 96  .z..^.|.o.].Kv..
    a7 ac 5a b4 00 00 00 00 00 00 00 00 00 00 00 00  ..Z.............
  backtrace:
    [<00000000762bb479>] __kmalloc+0x15b/0x250
    [<00000000aae191dc>] aa_calc_profile_hash+0x73/0x150
    [<0000000057e38ce8>] aa_unpack+0x38b/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de532c50c00 (size 64):
  comm "apparmor_parser", pid 660, jiffies 4294896613 (age 1654.372s)
  hex dump (first 32 bytes):
    73 6e 61 70 2e 63 6f 72 65 2e 38 39 33 35 2e 75  snap.core.8935.u
    73 72 2e 6c 69 62 2e 73 6e 61 70 64 2e 73 6e 61  sr.lib.snapd.sna
  backtrace:
    [<00000000762bb479>] __kmalloc+0x15b/0x250
    [<00000000830d06cf>] __aafs_profile_mkdir+0x28a/0x320
    [<00000000f8a3f998>] aa_replace_profiles+0x525/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
root@husky-virtual-machine:/#
```

Figure 6: Kmemleak showing 2 data leaks during the 73 hours


Activities
Terminal
Player
Ubuntu 64-bit copy
root@husky-virtual-machine: /
File Edit View Search Terminal Help

```
unreferenced object 0xffff9de53530f000 (size 1024):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.164s)
  hex dump (first 32 bytes):
    44 01 c5 32 e5 9d ff ff 44 01 c5 32 e5 9d ff ff  D..2....D..2....
    10 ac 30 35 e5 9d ff ff 00 02 00 00 00 00 ad de  ..05............
  backtrace:
    [<00000000ea28a401>] kmem_cache_alloc_trace+0xf8/0x1e0
    [<00000000ead0e84c>] aa_alloc_profile+0x3b/0xf0
    [<000000006f3c6290>] unpack_profile+0x133/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de532a6c360 (size 96):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.236s)
  hex dump (first 32 bytes):
    01 00 00 00 00 00 00 00 00 00 2b 33 e5 9d ff ff  ..........+3....
    00 a0 d0 32 e5 9d ff ff 00 e0 d0 32 e5 9d ff ff  ...2.......2....
  backtrace:
    [<00000000ea28a401>] kmem_cache_alloc_trace+0xf8/0x1e0
    [<00000000a1b31676>] aa_dfa_unpack+0x34/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<00000000436307f5>] unpack_profile+0x5f4/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
```

Figure 7: Kmemleak showing the next 2 data leaks during the 73 hours


Activities
Terminal
Player
Ubuntu 64-bit copy
root@husky-virtual-machine: /
File Edit View Search Terminal Help

```
unreferenced object 0xffff9de532d08000 (size 8192):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.272s)
  hex dump (first 32 bytes):
    07 00 02 00 00 00 00 00 41 08 00 00 00 00 10 00  ........A.......
    02 00 1e 00 02 00 11 00 2b 00 03 00 0c 00 04 00  ........+.......
  backtrace:
    [<00000000f0cffb1b>] __kmalloc_node+0x1a9/0x2f0
    [<00000000c0f6b299>] kvmalloc_node+0x4b/0x80
    [<00000000c8ae49a2>] aa_dfa_unpack+0x14e/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<00000000436307f5>] unpack_profile+0x5f4/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de532a6c6c0 (size 96):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.308s)
  hex dump (first 32 bytes):
    01 00 00 00 00 00 00 00 00 c0 d0 32 e5 9d ff ff  ...........2....
    00 20 45 34 e5 9d ff ff 00 40 06 34 e5 9d ff ff  . E4.....@.4....
  backtrace:
    [<00000000ea28a401>] kmem_cache_alloc_trace+0xf8/0x1e0
    [<00000000a1b31676>] aa_dfa_unpack+0x34/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<0000000014ea0bd8>] unpack_profile+0x505/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
```

Figure 8: Kmemleak showing another 2 data leaks during the 73 hours

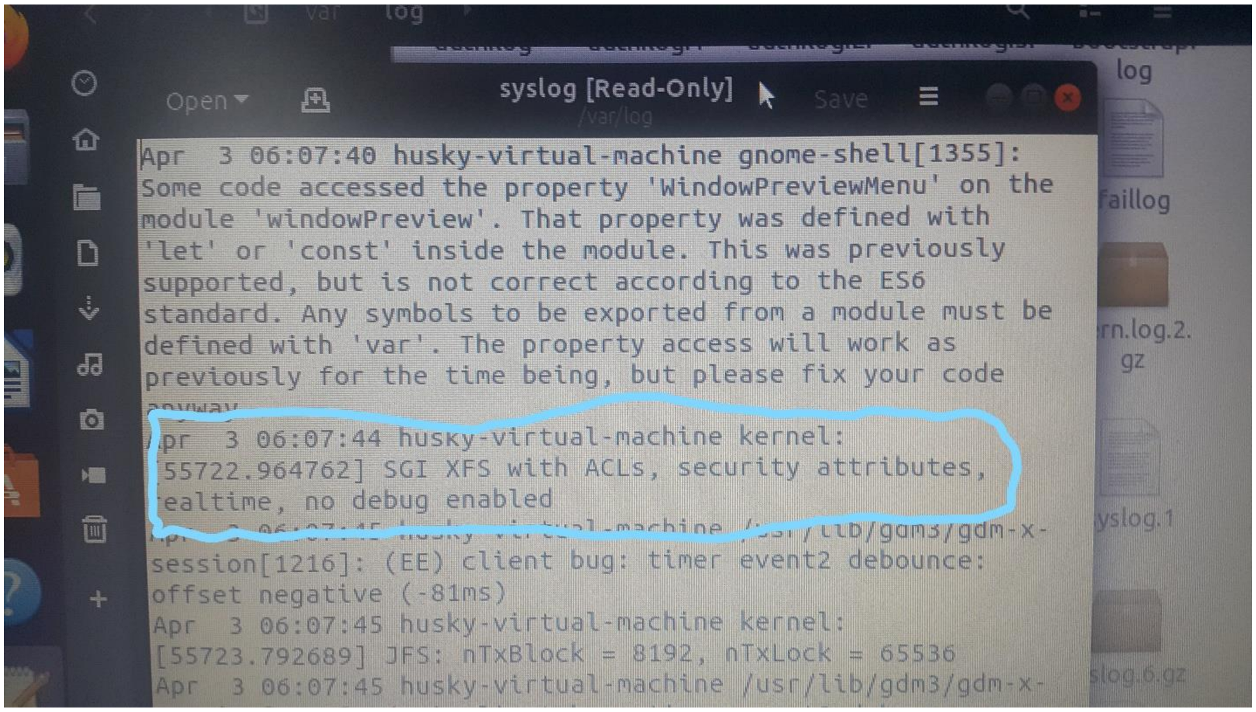
Data Analysis

To perform data analysis, the plan is to collect data continuously for 5 days. However, referring to table 2 on page 35, 71 hours later, the performance of the computer degraded to a point of where the system stopped responding and needed a reboot. When the system stopped responding, the network devices could not send any more event messages to syslog. Therefore, syslog did not record anymore events of kernel memory leakage. However, after a system reboot, the syslog did show a total number of 30 new suspected memory leaks. This brings to light that an attacker can modify the syslog file to avoid been detected (O’Reilly Media, Inc., 2020). Fortunately, kmemleak was re-compile with the Linux kernel and not an LKM. That is the reason why kmemleak was still operating and showed kernel memory leakage that was not reported in syslog before the reboot. This means that the syslog needs to be protected and if the system stopped responding, kmemleak needs to be analyzed for kernel memory leakage.

Table 2: Data Analysis of kernel memory leakage

Date	Mar 29	Mar 30		Mar 31	Apr 1	Apr 2	Apr 3
Time (am)	9:07	8:03	8:13	8:13	8:13	9:19	11:05
Hours monitoring kernel leaks	23 hours			73 hours			26 hours
	71 hours						
	96 hours						
	122 hours						
Number of leaks	14 leaks			16 leaks			N/A
	30 leaks						N/A
Number of times scan	138 times			Unknown number of times scan			N/A
				System stopped responding			
Event		A					

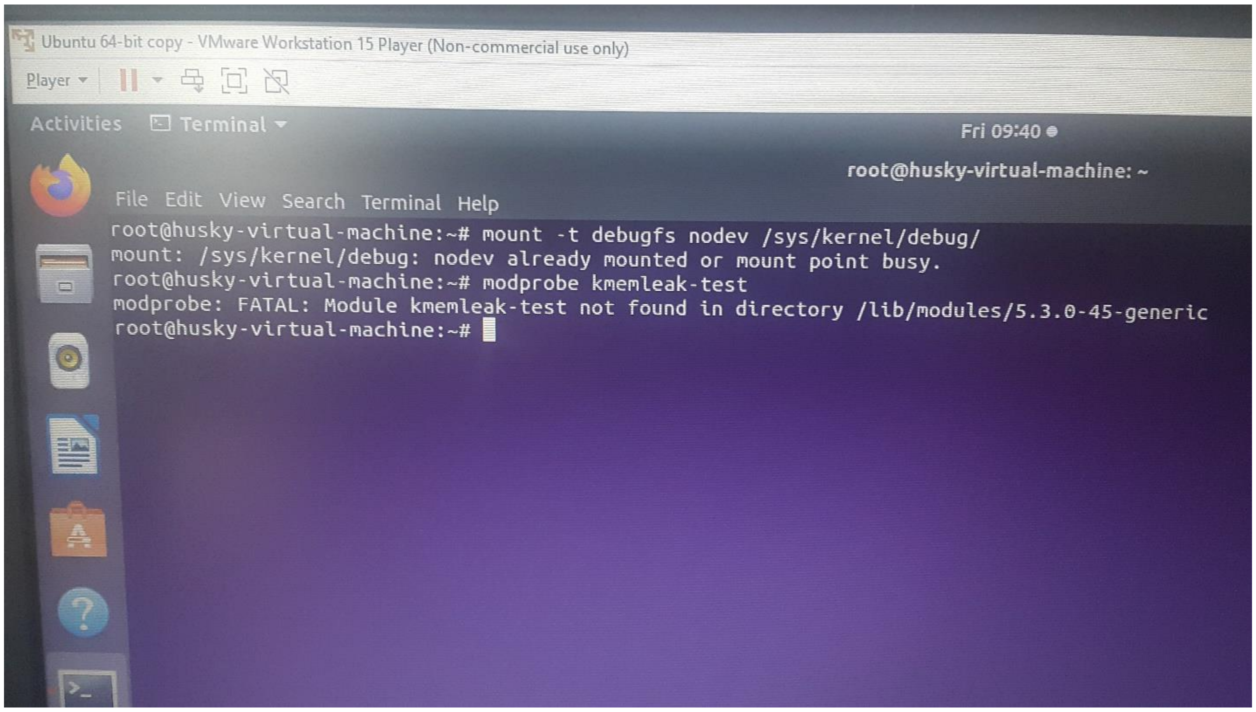
Based on the syslog records in figure 3, 4 and 5 on page 29 and 30, 14 new suspected memory leaks were found. By default, kmemleak will scan for kernel memory leakage is performed every 10 minutes and the even is recorded in syslog. For the first data collection, the total time monitored for kernel memory leakage is 23 hours. Based on the 10 minutes interval of scanning, the memory is scan 6 times in an hour and multiply by 23 hours will equal to 138 times of scan performed for the total duration. During this fata collection time, 14 new suspected leaks were found. That is approximately 10% of leakage based on the number of scans compared with the number of suspected new leaks for the system that is been monitored. In the following 73 hours when the syslog stopped responding, there were 16 kernel memory leakage found. 6 leaks had arbitrary data in the memory leak and 10 leaks did not had any data in the memory leak. The calculation for the output during the 73 hours will not be measured against the first 23 hours monitored as the syslog had stopped responding. The output from the 73 hours of monitoring shows that 37.5% percent of the data leaked from kernel memory contains arbitrary data while the remaining data leaked that does not show any data is 62.5% percent as shown in figure 6, 7 and 8 on page 33 and 34. Though syslog has stopped responding and both the 23 hours and 73 hours data collection period cannot be compared based on the number of scans, the measurement of total time and number of leaks can still be compared. At this time referring to table 2 on page 35, the total time of kernel memory monitored is 96 hours with a total of 30 leaks. That is, on the average 1 leak in every 3.2 hours of the time monitored. Therefore, this computer system average leak of kernel data in the first 23 hours is 1.4 leak in every 2.3 hours compare to the total 96 hours with 1 leak in every 3.2 hours. This shows that from the beginning of the data monitoring collected, the number of leaks is decreasing as time increases. The last day of data collection did not show any data collected in the syslog. When a trigger was initiated to scan for memory leakage, permission was denied even though the scan was done with a root account. The computer was rebooted and after a closer look at the syslog, there is an entry from the kernel stating “no debug enabled” as shown in figure 9 on page 37.



```
Apr  3 06:07:40 husky-virtual-machine gnome-shell[1355]:
Some code accessed the property 'WindowPreviewMenu' on the
module 'windowPreview'. That property was defined with
'let' or 'const' inside the module. This was previously
supported, but is not correct according to the ES6
standard. Any symbols to be exported from a module must be
defined with 'var'. The property access will work as
previously for the time being, but please fix your code
anyway
Apr  3 06:07:44 husky-virtual-machine kernel:
[55722.964762] SGI XFS with ACLs, security attributes,
realtime, no debug enabled
Apr  3 06:07:45 husky-virtual-machine /usr/lib/gdm3/gdm-x-
session[1216]: (EE) client bug: timer event2 debounce:
offset negative (-81ms)
Apr  3 06:07:45 husky-virtual-machine kernel:
[55723.792689] JFS: nTxBlock = 8192, nTxLock = 65536
Apr  3 06:07:45 husky-virtual-machine /usr/lib/gdm3/gdm-x-
```

Figure 9: Syslog showing “no debug enabled”

A command was executed to mount the debug and the mount point is already mounted or mount point busy as shown figure 10 on page 38. Then a test was done to deliberately cause a leak in kernel memory to test if kmemleak is working. The system response is “FATAL: Module kmemleak-test not found in directory” as shown in figure 10 on page 38. The kmemleak file itself is missing from the /sys/kernel/debug/ directory.


Ubuntu 64-bit copy - VMware Workstation 15 Player (Non-commercial use only)
Player
Activities
Terminal
Fri 09:40
root@husky-virtual-machine: ~
File Edit View Search Terminal Help

```
root@husky-virtual-machine:~# mount -t debugfs nodev /sys/kernel/debug/
mount: /sys/kernel/debug: nodev already mounted or mount point busy.
root@husky-virtual-machine:~# modprobe kmemleak-test
modprobe: FATAL: Module kmemleak-test not found in directory /lib/modules/5.3.0-45-generic
root@husky-virtual-machine:~#
```

Figure 10: Result of mounting kmemleak and kmemleak-test

Therefore, no data is available for the last day of data collection. The investigation done leads to an understanding that kmemleak has crashed after running non-stop for 73 hours. In addition, as the mount point is already mounted, the kmemleak will automatically load itself as shown in table 2 on page 35 listed as “Event” at the bottom of the table indicated with the letter “A” on Mar 30 at 8:03am. However, of course this autoloading is before the crash.

Summary

In conclusion, the data of kernel memory leakage that was collected for 4 days concludes that the kernel memory leakage did reduce from 1.4 leak in every 2.3 hours to 1 leak in every 3.2 hours for this computer system. As for the purpose of this study, the kernel memory leakage can be read and recorded in syslog file format and a text file format for the actual data leaked from kernel memory.

Chapter V: Results, Conclusion, and Recommendations

Introduction

After designing and testing the methodology, and presenting and analyzing the data, this chapter concludes the findings and recommends the future work.

Results

To address the first study question at the beginning of this study, that is how can the kernel memory be read? There are two methods available to access the kernel memory. The first method is to use Loadable Kernel Module (LKM) that can be loaded and unloaded as needed. However, if the LKM is unloaded without removing the changes made, the kernel memory will be locked until a reboot takes place. That is why kernel developers need to know the exact location of the memory and what was done to the memory as there is no Garbage Collector (GC) to clean up after them. Otherwise the LKM will be the reason why there is a kernel memory leakage (Day, 2009). The second method to access the kernel memory is to re-compile the Linux kernel image. This method is better as the kernel memory leakage detector, kmemleak is in the kernel and prove to be an advantage for this study because when kmemleak crashed, the syslog was updated to reflect the 30 new suspected memory leakage. LKM may not have reflected these changes after the system was rebooted. The second study question addresses the concern of, if there is any log file created from the scanning of kernel memory leakage? There are 2 log files created, one is from syslog that records the event messages from the network devices. The other log file is from kmemleak that shows the actual data leaked from kernel memory. The final question is concern over the readability of the log files by a Host-based IDS? The answer to this question is yes, a Host-based IDS can read a syslog file format and capture the details from the syslog file based on the program that generated the event message.

Conclusion

This study is a proof-of-concept to address the need of having a Host-based IDS to promote an alert system for side-channel attacks such as Meltdown and Spectre. To complete this study, scripts were written to facilitate an improved environment where the kernel memory leak detector can be installed with only a few inputs needed from the user such as the root account password and selection to confirm the installation. Analysis was done to compare the leaked data collected from the kernel memory over a period of 5 days showing a reduction in kernel memory leakage detected over a longer time period. Therefore, kernel memory leakage can be read, and alerts created for the Host-based IDS.

As a result of identifying and reviewing that there is or are kernel memory leakage in the Host-based IDS, the Linux program out-of-memory killer (OOM) or kmemleak free internal objects is an option free up more kernel memory space to mitigate the risk of an attack or use. In conclusion, the solution to use a Host-based IDS to monitor and alert side-channel attacks related to kernel memory leakage is available. However, the technology is not integrated into one software that is needed to mitigate the risk of side-channel attacks. Hopefully this solution will be in the real-world soon as the current attack vector is not just data breach from a database, data in transmission, infrastructure, and web applications, but it includes data in memory.

Future Work

The main challenge faced by this study is the huge amount of log files created at every 10 minutes interval (The kernel development community, n.d.a). This can mean importing the data into a database server for storage and analysis. A data analysis study can then be performed on the log files to determine if the arbitrary data has any pattern or caused by specific process? This data analysis study will provide a method to narrow down the specific part of the kernel that is generating the arbitrary data leak. For example, is there a specific LKM that did not undo the changes when the LKM is unmounted? The second suggested future work is to identify the Indicators of Attack (IoA) to find out the conditions that are needed to detect when confidential data in the kernel memory is leaked. After the data analysis is performed and IoA are identified, future work for the Host-based IDS to develop a filtering rule based on anomalies to mitigate zero-day attacks. As more IoA are developed, the Host-based IDS will be more accurate in identifying kernel memory attacks to reduce the attack vectors for kernel memory. Another possible study is how to integrate kmemleak or LKM with a Host-based IDS that will result in a faster monitoring and alert system. The next future work is related to kmemleak where further research is needed to determine if date and time stamp can be included in kmemleak output. In addition, the results and methodologies for this study to develop a Host-based IDS as an alert system, can provide additional information for other security tools. For example, antivirus or malware program to identify zero-day threats in the kernel memory exploitations. The fifth future work is to enable kernel memory leakage monitoring for mobile phones. This is important for mobile phone has limited resources and the operating system does leak kernel memory (Talexop, 2020).

References

Canella, C., Bulck, J. V., Schwarz, M., Lipp, M., Berg, B. V., Ortner, P., . . . College of William and Mary. (2019, May 15). *ASystematicEvaluationofTransientExecutionAttacksandDefenses.* Retrieved from ASystematicEvaluationofTransientExecutionAttacksandDefenses: https://arxiv.org/pdf/1811.05441.pdf

Abrams, L. (2018, January 3). *List of Meltdown and Spectre Vulnerability Advisories, Patches, & Updates*. Retrieved from List of Meltdown and Spectre Vulnerability Advisories, Patches, & Updates: https://www.bleepingcomputer.com/news/security/list-of-meltdown-and-spectre-vulnerability-advisories-patches-and-updates/

ArchWiki. (2020, January 18). *Kernel module*. Retrieved from Kernel module: https://wiki.archlinux.org/index.php/Kernel_module

Clinton, D. (2018, May 30). *How to load or unload a Linux kernel module*. Retrieved from Set up a Tor proxy with Raspberry Pi to control internet traffic: https://opensource.com/article/18/5/how-load-or-unload-linux-kernel-module

Corbet. (2006, June 19). *Detecting kernel memory leaks*. Retrieved from Detecting kernel memory leaks: https://lwn.net/Articles/187979/

Day, R. (2009, July 8). *The Kernel Newbie Corner: Loadable Kernel Modules, Coming And Going*. Retrieved from https://www.linux.com: https://www.linux.com/training-tutorials/kernel-newbie-corner-loadable-kernel-modules-coming-and-going/

Graz University of Technology. (2018a). *Meltdown and Spectre*. Retrieved from Meltdown and Spectre: https://meltdownattack.com/

Graz University of Technology. (2018b). *Meltdown and Spectre*. Retrieved from Meltdown and Spectre: https://meltdownattack.com/

Isohara, T., Takemori, K., Miyake, Y., Qu, N., & Perrig, A. (2010). LSM-Based Secure System Monitoring Using Kernel Protection Schemes. *2010 International Conference on Availability, Reliability and Security* (p. 286). Krakow, Poland: IEEE.

Kerner, S. M. (2018, September 10). *Protecting Against the 7 Vulnerabilities of Meltdown and Spectre*. Retrieved from Protecting Against the 7 Vulnerabilities of

Meltdown and Spectre: https://www.esecurityplanet.com/applications/meltdown-and-spectre-vulnerabilities.html

Lhotsky, B. (2013). *OSSEC Host-based Intrusion Detection.* July: Packt Publishing Limited.

McAfee. (2014, October). *Indicators of Attack (IoA).* Retrieved from Solution Brief: https://www.mcafee.com/enterprise/en-us/assets/solution-briefs/sb-indicators-of-attack.pdf

MITRE Corporation. (2015 - 2020). *Kernel Modules and Extensions*. Retrieved from MITRE Att&ck: https://attack.mitre.org/techniques/T1215/

Molloy, D. (2015). *Writing a Linux Kernel Module — Part 1: Introduction*. Retrieved from Writing a Linux Kernel Module — Part 1: Introduction: http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/

Murali, R. (2013, November 25). *Os concepts.* Retrieved from Featured SlideShares: https://www.slideshare.net/SUDHEESHPENATHU/os-concepts-28589673

Ning, P., Qing, S., & Li, N. (2006). Information and Communications Security: 8th International Conference, ICICS 2006. *8th International Conference, ICICS 2006* (p. 182). Raleigh: Springer.

NIST. (2019, October 8). *NATIONAL VULNERABILITY DATABASE*. Retrieved from NATIONAL VULNERABILITY DATABASE: https://nvd.nist.gov/vuln/detail/CVE-2017-5754

O'Reilly Media, Inc. (2020). *Protect Your Logs from Tampering.* Retrieved from Protect Your Logs from Tampering: https://www.oreilly.com/library/view/network-security-hacks/0596006438/ch01s06.html

OSSEC Project Team. (2010-2020). *Log monitoring/analysis*. Retrieved from OSSEC is a scalable, multi-platform, open source Host-based Intrusion Detection System (HIDS): https://www.ossec.net/docs/docs/manual/monitoring/index.html

OSSEC PROJECT TEAM. (2020). *OSSEC is a scalable, multi-platform, open source Host-based Intrusion Detection System (HIDS)*. Retrieved from OSSEC is a scalable, multi-platform, open source Host-based Intrusion Detection System (HIDS): https://www.ossec.net/about/

Phrack Inc. (n.d.). *Infecting loadable kernel modules.* Retrieved from Phrack News: http://www.phrack.org/archives/issues/68/11.txt

Phrack Magazine. (2003, August 13). *Infecting Loadable Kernel Modules*. Retrieved from PHRACK NEWS: http://phrack.org/issues/61/10.html

Sanders, J. (2019, May 19). *Spectre and Meltdown explained: A comprehensive guide for professionals*. Retrieved from Spectre and Meltdown explained: A comprehensive guide for professionals: https://www.techrepublic.com/article/spectre-and-meltdown-explained-a-comprehensive-guide-for-professionals/

Schwartz, M. J. (2018, May 22). *Spectre and Meltdown Flaws: Two More Variants Discovered*. Retrieved from Bank Info Security: https://www.bankinfosecurity.com/spectre-meltdown-flaws-two-more-variants-discovered-a-11021

Schwarz, M., Canella, C., Giner, L., & Gruss, D. (2019.a). Store-to-Leak Forwarding: LeakingDataonMeltdown-resistantCPUs. *ArXiv*, 1.

Schwarz, M., Canella, C., Giner, L., & Gruss, D. (2019.b). Store-to-LeakForwarding: LeakingDataonMeltdown-resistantCPUs. *ArXiv 2019*, 1.

Silberschatz, a., Galvin, P. B., & Gagne, G. (2012). Solutions to Practice Exercises of the Ninth Edition of Operating System Concepts. In a. Silberschatz, P. B. Galvin, & G. Gagne, *Solutions to Practice Exercises of the Ninth Edition of Operating System Concepts* (p. 61). New Haven: John Wiley & Sons, Inc.

Smalley, S., Fraser, T., & Vance, C. (n.d.). *Linux Security Modules: General Security Hooks for Linux*. Retrieved from The Linux Kernel documentation: https://www.kernel.org/doc/html/latest/security/lsm.html

Talexop. (2020, February 2). *Android 10 - kernel memory leak bug*. Retrieved from Andrroid Develoopment and Hacking: https://forum.xda-developers.com/galaxy-note-10+/help/android-10-kernel-memory-leak-bug-t4044415

Technology, G. U. (2018). *Meltdown and Spectre*. Retrieved from Meltdown and Spectre: https://meltdownattack.com

The kernel development community. (2017, September 5). *Development tools for the Kernel .* Retrieved from Development tools for the Kernel : https://www.scribd.com/document/382273976/Kernel-Doc-Guide

The kernel development community. (n.d.a). *Kernel Memory Leak Detector*. Retrieved from The Linux Kernel documentation: https://www.kernel.org/doc/html/latest/dev-tools/kmemleak.html

The kernel development community. (n.d.b). *Kernel Memory Leak Detector*. Retrieved from The Linux Kernel documentation: https://www.kernel.org/doc/html/latest/dev-tools/kmemleak.html

The kernel development community. (n.d.). *The Linux Kernel documentation*. Retrieved from The Linux Kernel documentation: https://www.kernel.org/doc/

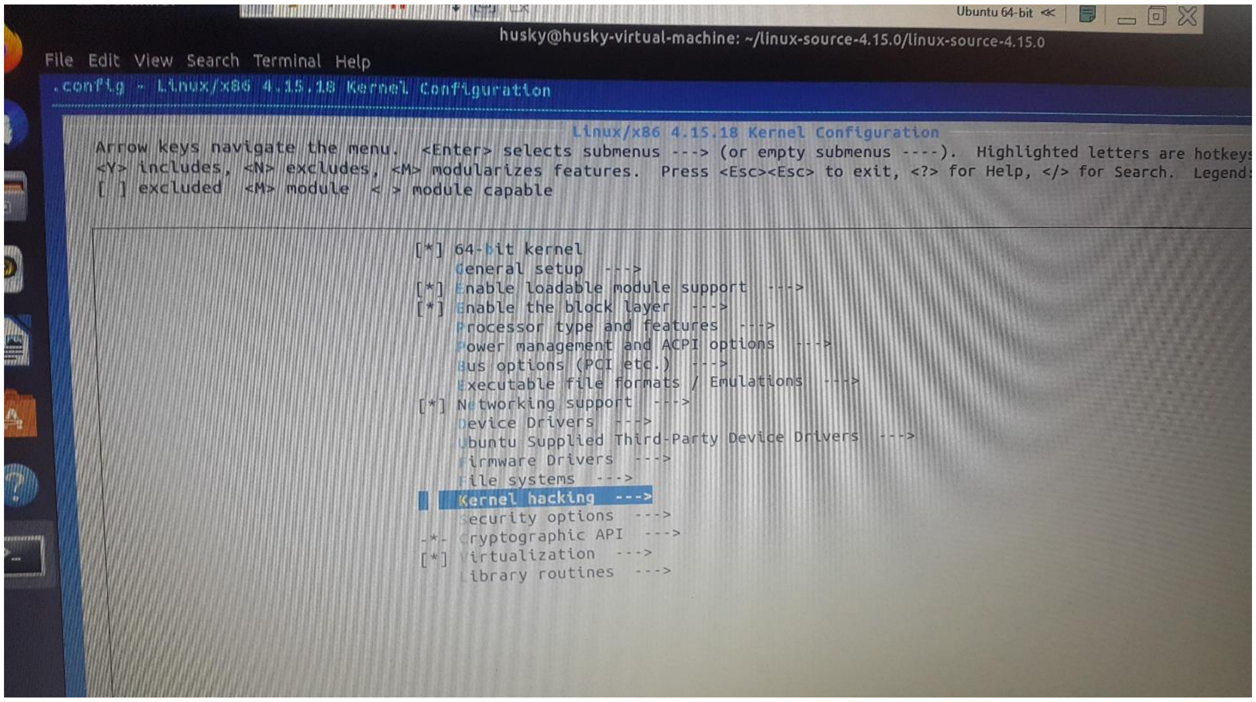
The MITRE Corporation . (2014). *SLEAK: A Side-channel Leakage Evaluator and Analysis Kit.* Retrieved from SLEAK: A Side-channel Leakage Evaluator and Analysis Kit: https://www.mitre.org/sites/default/files/publications/pr-14-3463-sleak-side-channel-leakage-evaluator.pdf

Threat Stack, Inc. (2015, September 9). *The History of Intrusion Detection Systems (IDS) – Part 1*. Retrieved from CLOUD SECURITY AND COMPLIANCE: https://www.threatstack.com/blog/the-history-of-intrusion-detection-systems-ids-part-1

Trend Micro. (2019). *Indicators of Compromise*. Retrieved from Prepare for, withstand, and rapidly recover from threats - now and in the future. : https://www.trendmicro.com/vinfo/us/security/definition/indicators-of-compromise

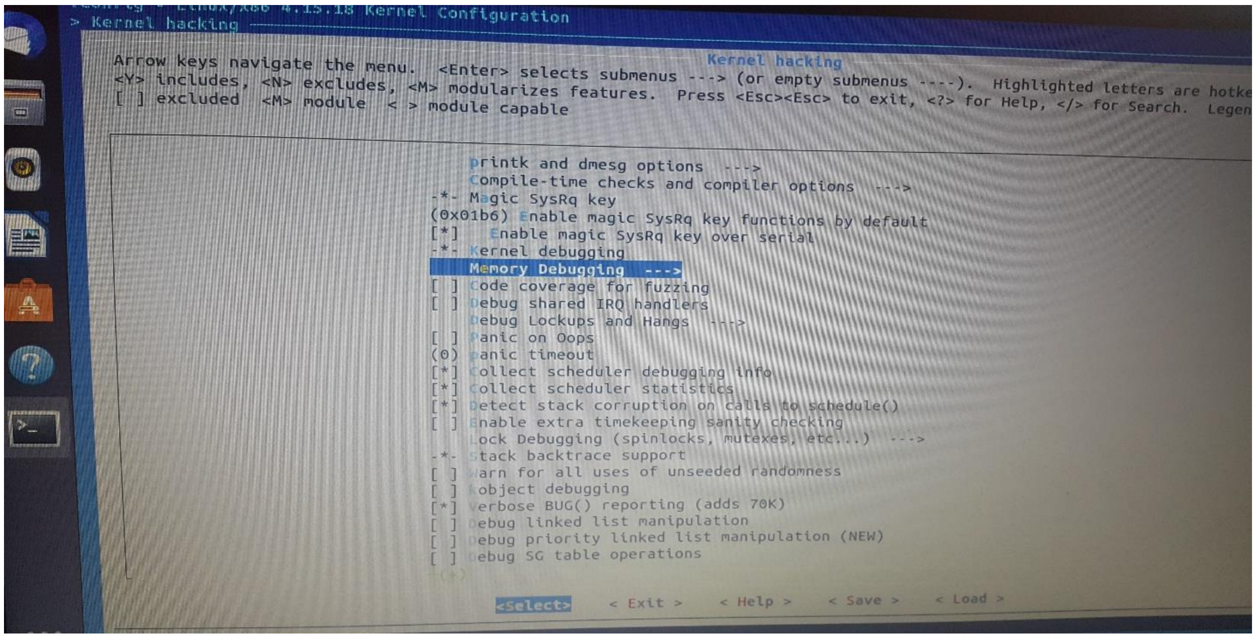
Walters, D., Hagen, A., & Kedaigle, E. (2014). *SLEAK: A Side-channel Leakage Evaluator and Analysis Kit.* Retrieved from SLEAK: A Side-channel Leakage Evaluator and Analysis Kit: www.mitre.org

Appendix A: Additional Information



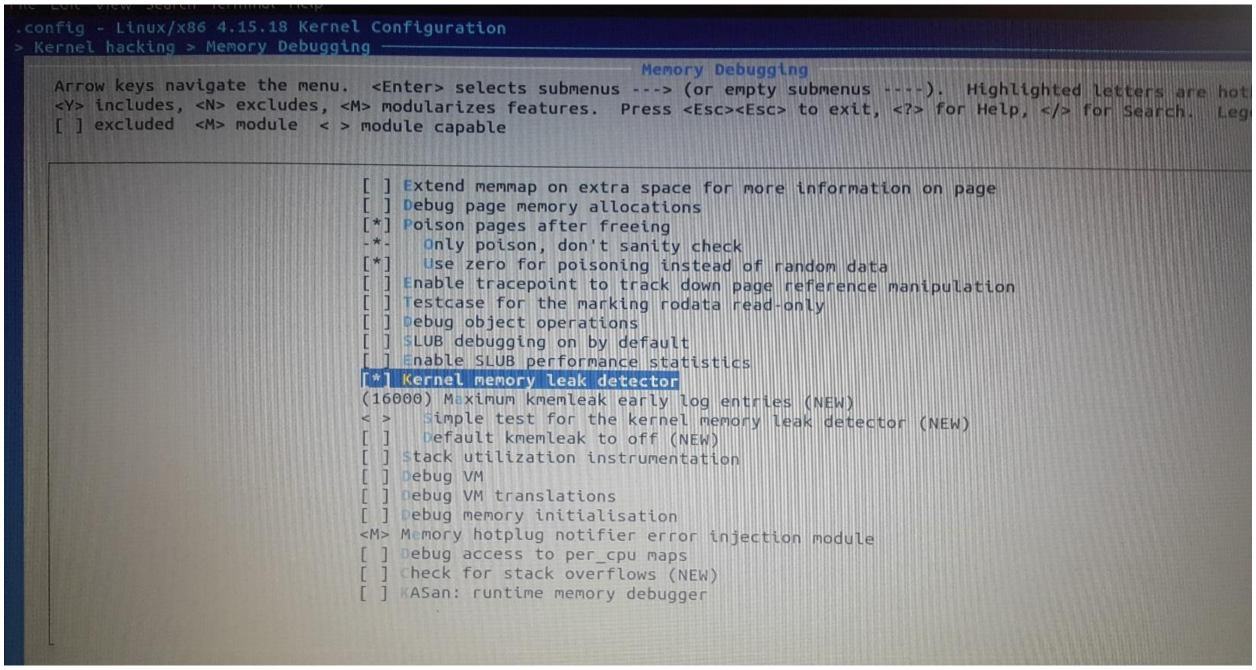
```
husky@husky-virtual-machine: ~/linux-source-4.15.0/linux-source-4.15.0
File  Edit  View  Search  Terminal  Help
.config - Linux/x86 4.15.18 Kernel Configuration

                    Linux/x86 4.15.18 Kernel Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys
<Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend:
[ ] excluded  <M> module  < > module capable

              [*] 64-bit kernel
                  General setup  --->
              [*] Enable loadable module support  --->
              [*] Enable the block layer  --->
                  Processor type and features  --->
                  Power management and ACPI options  --->
                  Bus options (PCI etc.)  --->
                  Executable file formats / Emulations  --->
              [*] Networking support  --->
                  Device Drivers  --->
                  Ubuntu Supplied Third-Party Device Drivers  --->
                  Firmware Drivers  --->
                  File systems  --->
                  Kernel hacking  --->
                  Security options  --->
              -*- Cryptographic API  --->
              [*] Virtualization  --->
                  Library routines  --->
```

Figure 11: Select module kernel hacking in Configure Kernel using “make menuconfig”.



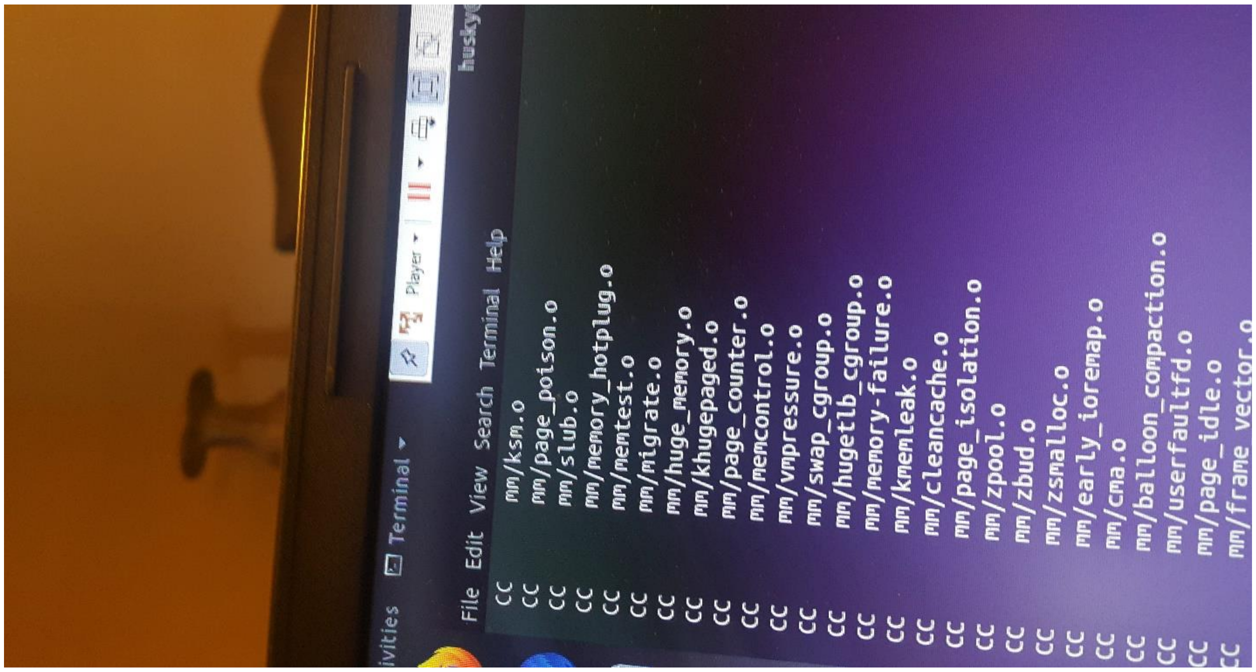
```
.config - Linux/x86 4.15.18 Kernel Configuration
> Kernel hacking
                              Kernel hacking
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotke
<Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legen
[ ] excluded  <M> module  < > module capable

                  printk and dmesg options  --->
                  Compile-time checks and compiler options  --->
              -*- Magic SysRq key
              (0x01b6) Enable magic SysRq key functions by default
              [*]   Enable magic SysRq key over serial
              -*- Kernel debugging
                  Memory Debugging  --->
              [ ] Code coverage for fuzzing
              [ ] Debug shared IRQ handlers
                  Debug Lockups and Hangs  --->
              [ ] Panic on Oops
              (0) panic timeout
              [*] Collect scheduler debugging info
              [*] Collect scheduler statistics
              [*] Detect stack corruption on calls to schedule()
              [ ] Enable extra timekeeping sanity checking
                  Lock Debugging (spinlocks, mutexes, etc...)  --->
              -*- Stack backtrace support
              [ ] Warn for all uses of unseeded randomness
              [ ] kobject debugging
              [*] Verbose BUG() reporting (adds 70K)
              [ ] Debug linked list manipulation
              [ ] Debug priority linked list manipulation (NEW)
              [ ] Debug SG table operations
              ↓(+)

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 12: Selected Memory Debugging from Kernel Configuration



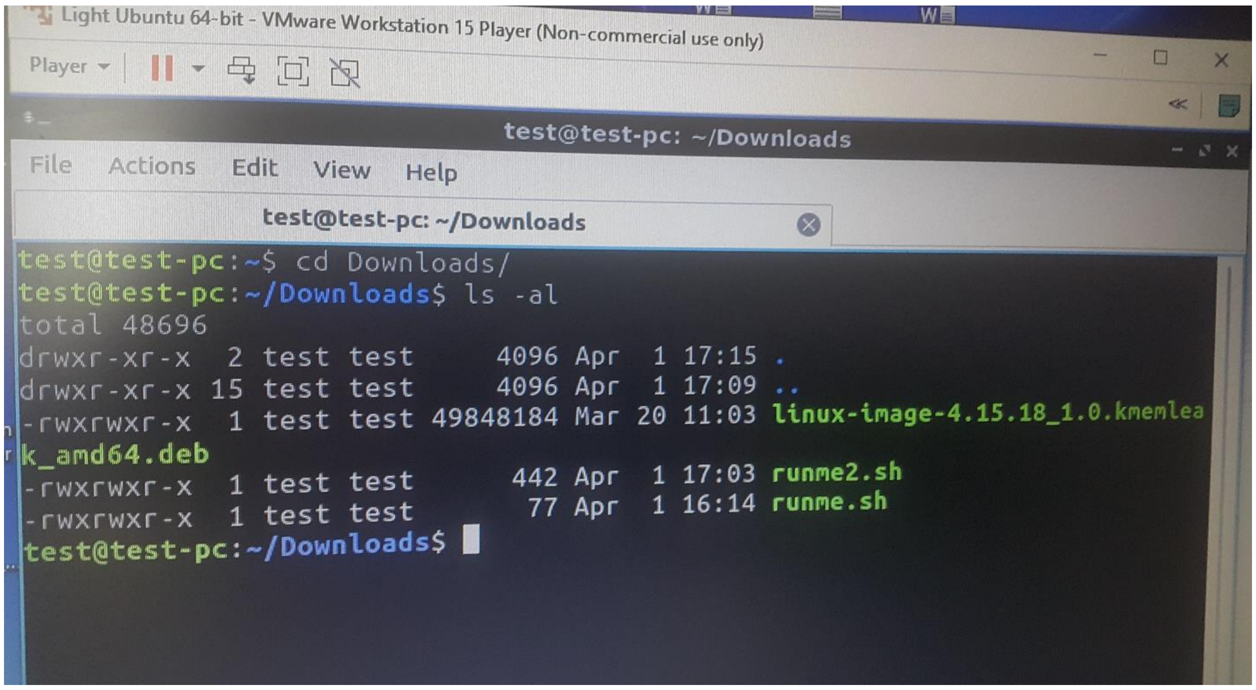
```
.config - Linux/x86 4.15.18 Kernel Configuration
> Kernel hacking > Memory Debugging
                                  Memory Debugging
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hot
  <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Leg
  [ ] excluded  <M> module  < > module capable

                          [ ] Extend memmap on extra space for more information on page
                          [ ] Debug page memory allocations
                          [*] Poison pages after freeing
                          -*-   Only poison, don't sanity check
                          [*]   Use zero for poisoning instead of random data
                          [ ] Enable tracepoint to track down page reference manipulation
                          [ ] Testcase for the marking rodata read-only
                          [ ] Debug object operations
                          [ ] SLUB debugging on by default
                          [ ] Enable SLUB performance statistics
                          [*] Kernel memory leak detector
                          (16000) Maximum kmemleak early log entries (NEW)
                          < >   Simple test for the kernel memory leak detector (NEW)
                          [ ]   Default kmemleak to off (NEW)
                          [ ] Stack utilization instrumentation
                          [ ] Debug VM
                          [ ] Debug VM translations
                          [ ] Debug memory initialisation
                          <M> Memory hotplug notifier error injection module
                          [ ] Debug access to per_cpu maps
                          [ ] Check for stack overflows (NEW)
                          [ ] KASan: runtime memory debugger
```

Figure 13: Enable the kernel memory leak detector



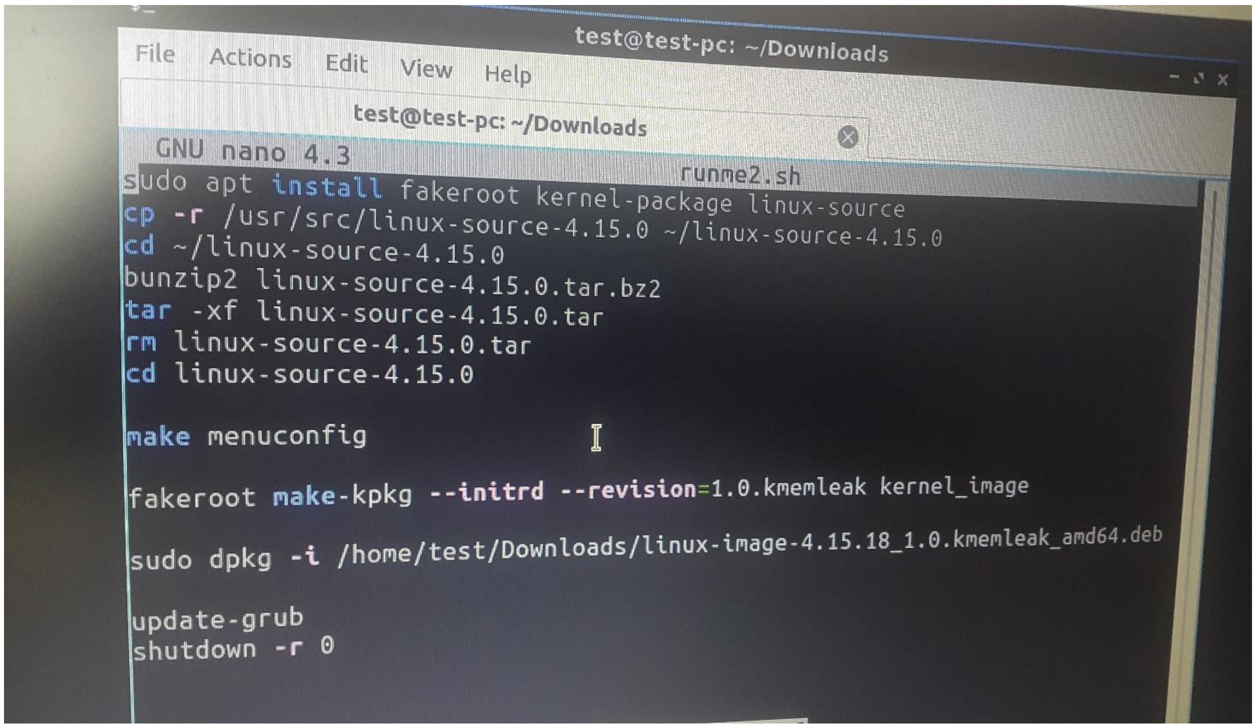
```
  CC      mm/ksm.o
  CC      mm/page_poison.o
  CC      mm/slub.o
  CC      mm/memory_hotplug.o
  CC      mm/memtest.o
  CC      mm/migrate.o
  CC      mm/huge_memory.o
  CC      mm/khugepaged.o
  CC      mm/page_counter.o
  CC      mm/memcontrol.o
  CC      mm/vmpressure.o
  CC      mm/swap_cgroup.o
  CC      mm/hugetlb_cgroup.o
  CC      mm/memory-failure.o
  CC      mm/kmemleak.o
  CC      mm/cleancache.o
  CC      mm/page_isolation.o
  CC      mm/zpool.o
  CC      mm/zbud.o
  CC      mm/zsmalloc.o
  CC      mm/early_ioremap.o
  CC      mm/cma.o
  CC      mm/balloon_compaction.o
  CC      mm/userfaultfd.o
  CC      mm/page_idle.o
  CC      mm/frame_vector.o
```

Figure 14: Rebuild Linux kernel modules from source


Light Ubuntu 64-bit - VMware Workstation 15 Player (Non-commercial use only)
Player
test@test-pc: ~/Downloads
File Actions Edit View Help

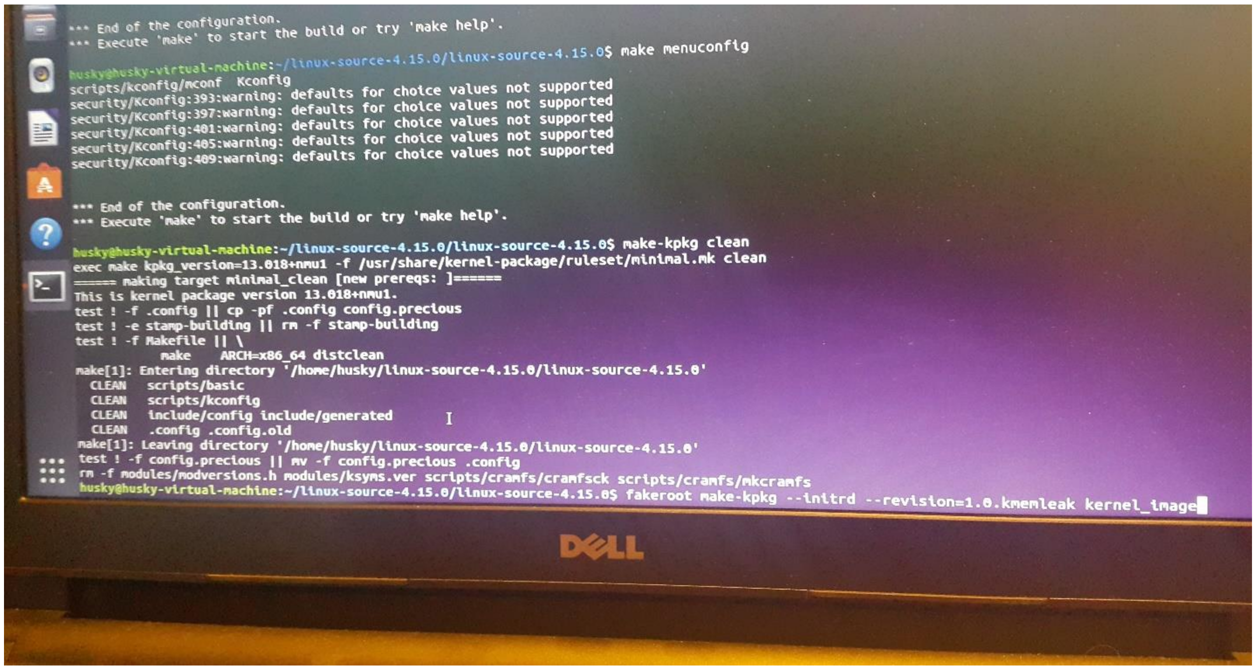
```
test@test-pc:~$ cd Downloads/
test@test-pc:~/Downloads$ ls -al
total 48696
drwxr-xr-x  2 test test     4096 Apr  1 17:15 .
drwxr-xr-x 15 test test     4096 Apr  1 17:09 ..
-rwxrwxr-x  1 test test 49848184 Mar 20 11:03 linux-image-4.15.18_1.0.kmemlea
k_amd64.deb
-rwxrwxr-x  1 test test      442 Apr  1 17:03 runme2.sh
-rwxrwxr-x  1 test test       77 Apr  1 16:14 runme.sh
test@test-pc:~/Downloads$
```

Figure 15: Bash script runme2.sh to install kmemleak



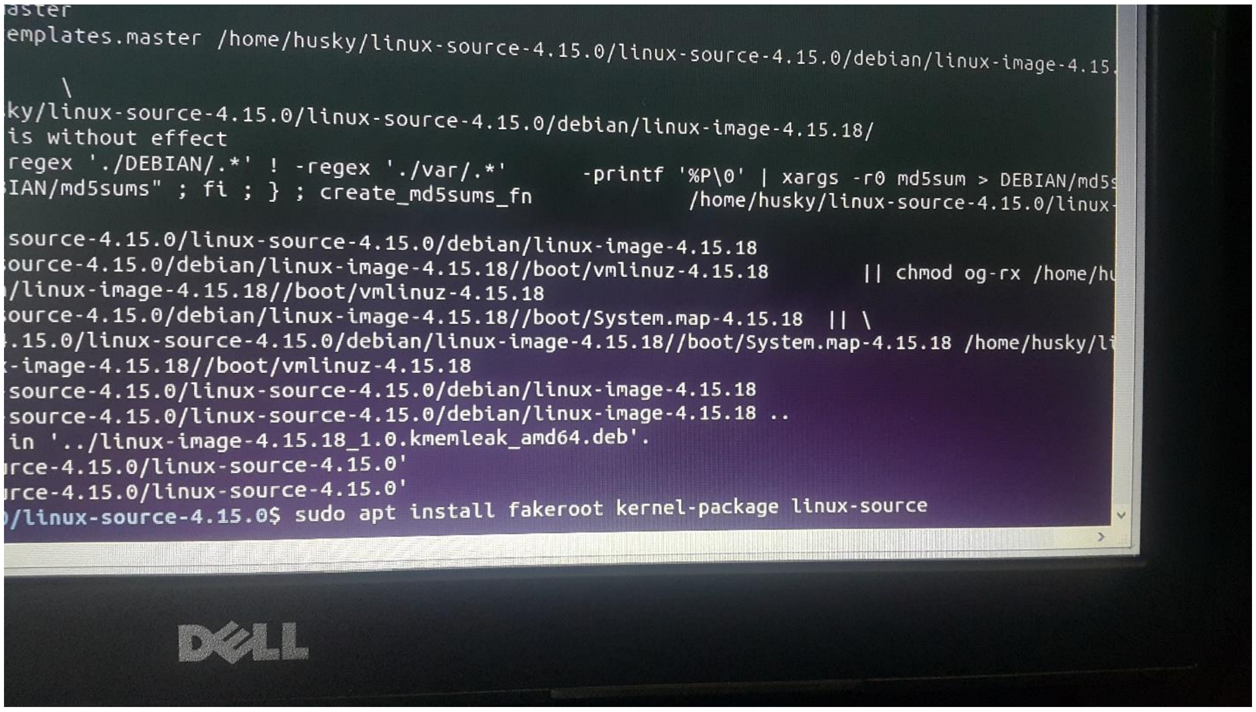
```
sudo apt install fakeroot kernel-package linux-source
cp -r /usr/src/linux-source-4.15.0 ~/linux-source-4.15.0
cd ~/linux-source-4.15.0
bunzip2 linux-source-4.15.0.tar.bz2
tar -xf linux-source-4.15.0.tar
rm linux-source-4.15.0.tar
cd linux-source-4.15.0

make menuconfig

fakeroot make-kpkg --initrd --revision=1.0.kmemleak kernel_image

sudo dpkg -i /home/test/Downloads/linux-image-4.15.18_1.0.kmemleak_amd64.deb

update-grub
shutdown -r 0
```

Figure 16: Content of bash script runme2.sh



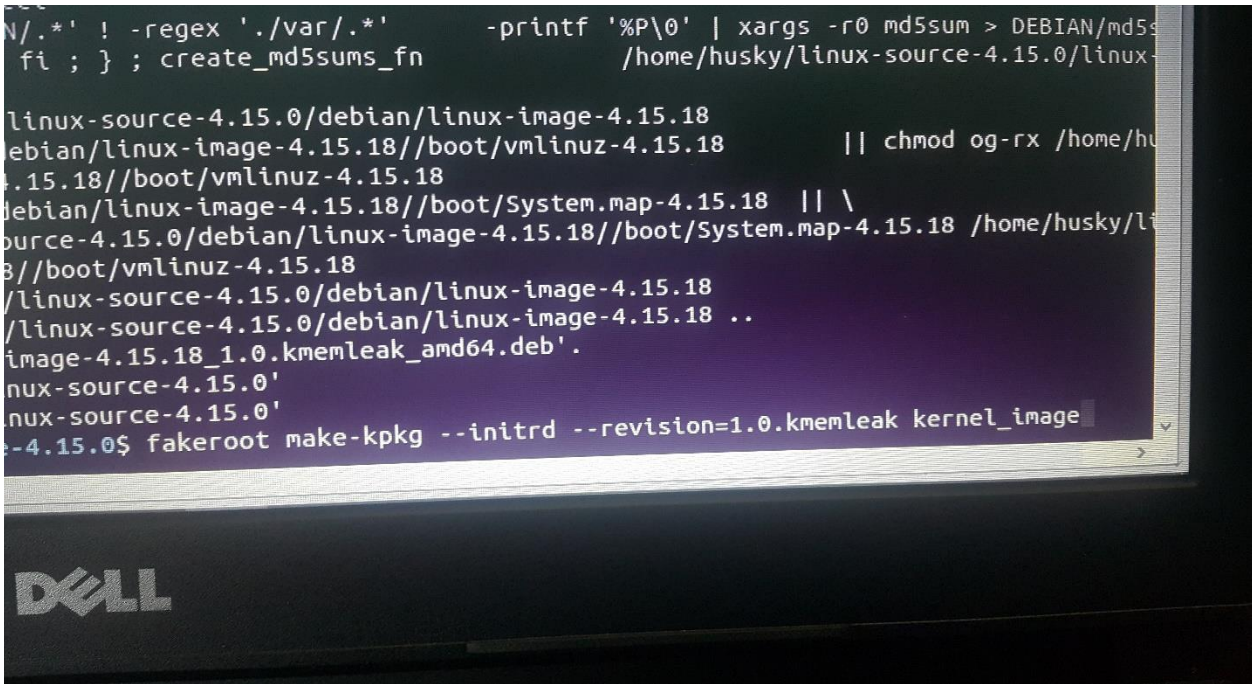
```
*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

husky@husky-virtual-machine:~/linux-source-4.15.0/linux-source-4.15.0$ make menuconfig
scripts/kconfig/mconf  Kconfig
security/Kconfig:393:warning: defaults for choice values not supported
security/Kconfig:397:warning: defaults for choice values not supported
security/Kconfig:401:warning: defaults for choice values not supported
security/Kconfig:405:warning: defaults for choice values not supported
security/Kconfig:409:warning: defaults for choice values not supported


*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

husky@husky-virtual-machine:~/linux-source-4.15.0/linux-source-4.15.0$ make-kpkg clean
exec make kpkg_version=13.018+nmu1 -f /usr/share/kernel-package/ruleset/minimal.mk clean
====== making target minimal_clean [new prereqs: ]======
This is kernel package version 13.018+nmu1.
test ! -f .config || cp -pf .config config.precious
test ! -e stamp-building || rm -f stamp-building
test ! -f Makefile || \
            make    ARCH=x86_64 distclean
make[1]: Entering directory '/home/husky/linux-source-4.15.0/linux-source-4.15.0'
  CLEAN   scripts/basic
  CLEAN   scripts/kconfig
  CLEAN   include/config include/generated
  CLEAN   .config .config.old
make[1]: Leaving directory '/home/husky/linux-source-4.15.0/linux-source-4.15.0'
test ! -f config.precious || mv -f config.precious .config
rm -f modules/modversions.h modules/ksyms.ver scripts/cramfs/cramfsck scripts/cramfs/mkcramfs
husky@husky-virtual-machine:~/linux-source-4.15.0/linux-source-4.15.0$ fakeroot make-kpkg --initrd --revision=1.0.kmemleak kernel_image
```

Figure 17: Allocated memory is low.



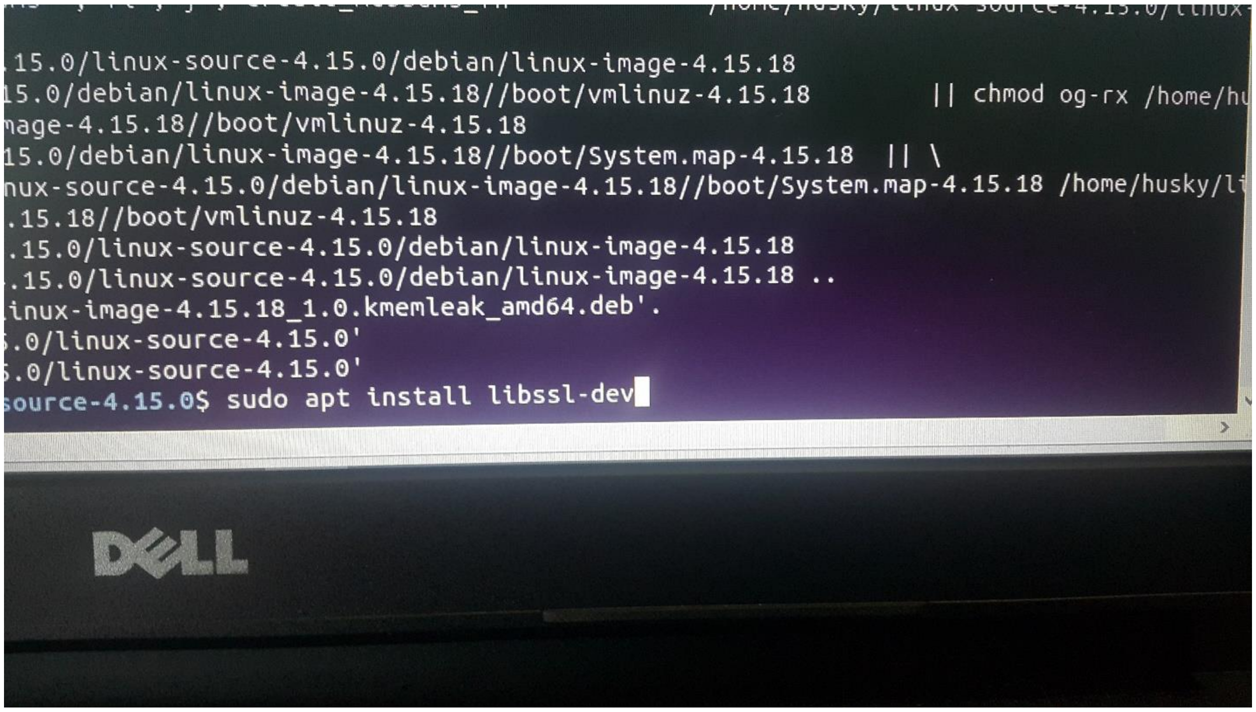
```
aster
emplates.master /home/husky/linux-source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.

   \
ky/linux-source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.18/
is without effect
regex './DEBIAN/.*' ! -regex './var/.*'      -printf '%P\0' | xargs -r0 md5sum > DEBIAN/md5s
IAN/md5sums" ; fi ; } ; create_md5sums_fn                  /home/husky/linux-source-4.15.0/linux-

source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.18
ource-4.15.0/debian/linux-image-4.15.18//boot/vmlinuz-4.15.18        || chmod og-rx /home/hu
/linux-image-4.15.18//boot/vmlinuz-4.15.18
ource-4.15.0/debian/linux-image-4.15.18//boot/System.map-4.15.18  || \
.15.0/linux-source-4.15.0/debian/linux-image-4.15.18//boot/System.map-4.15.18 /home/husky/li
-image-4.15.18//boot/vmlinuz-4.15.18
source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.18
source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.18 ..
in '../linux-image-4.15.18_1.0.kmemleak_amd64.deb'.
rce-4.15.0/linux-source-4.15.0'
rce-4.15.0/linux-source-4.15.0'
)/linux-source-4.15.0$ sudo apt install fakeroot kernel-package linux-source
```

Figure 18: Compile a new kernel package source



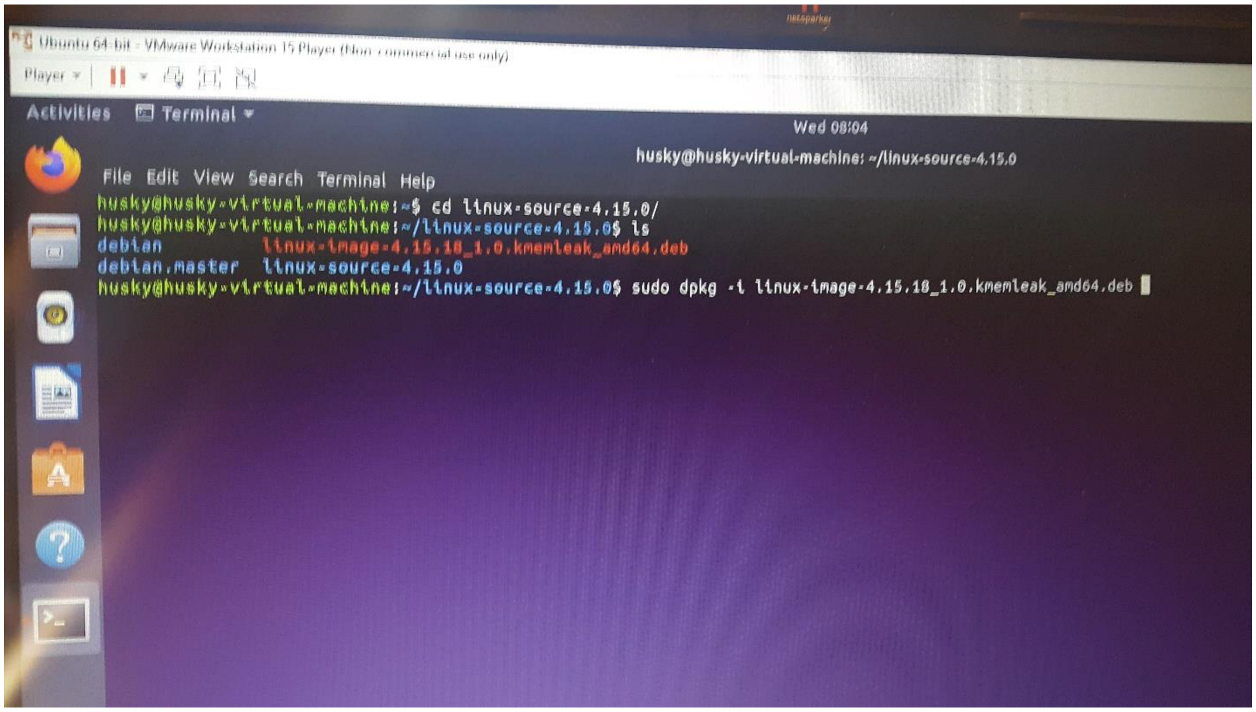
```
N/.*' ! -regex './var/.*'       -printf '%P\0' | xargs -r0 md5sum > DEBIAN/md5s
 fi ; } ; create_md5sums_fn             /home/husky/linux-source-4.15.0/linux-

linux-source-4.15.0/debian/linux-image-4.15.18
lebian/linux-image-4.15.18//boot/vmlinuz-4.15.18        || chmod og-rx /home/hu
.15.18//boot/vmlinuz-4.15.18
lebian/linux-image-4.15.18//boot/System.map-4.15.18  || \
ource-4.15.0/debian/linux-image-4.15.18//boot/System.map-4.15.18 /home/husky/li
3//boot/vmlinuz-4.15.18
/linux-source-4.15.0/debian/linux-image-4.15.18
/linux-source-4.15.0/debian/linux-image-4.15.18 ..
image-4.15.18_1.0.kmemleak_amd64.deb'.
nux-source-4.15.0'
nux-source-4.15.0'
-4.15.0$ fakeroot make-kpkg --initrd --revision=1.0.kmemleak kernel_image
```

Figure 19: Generate kmemleak kernel image



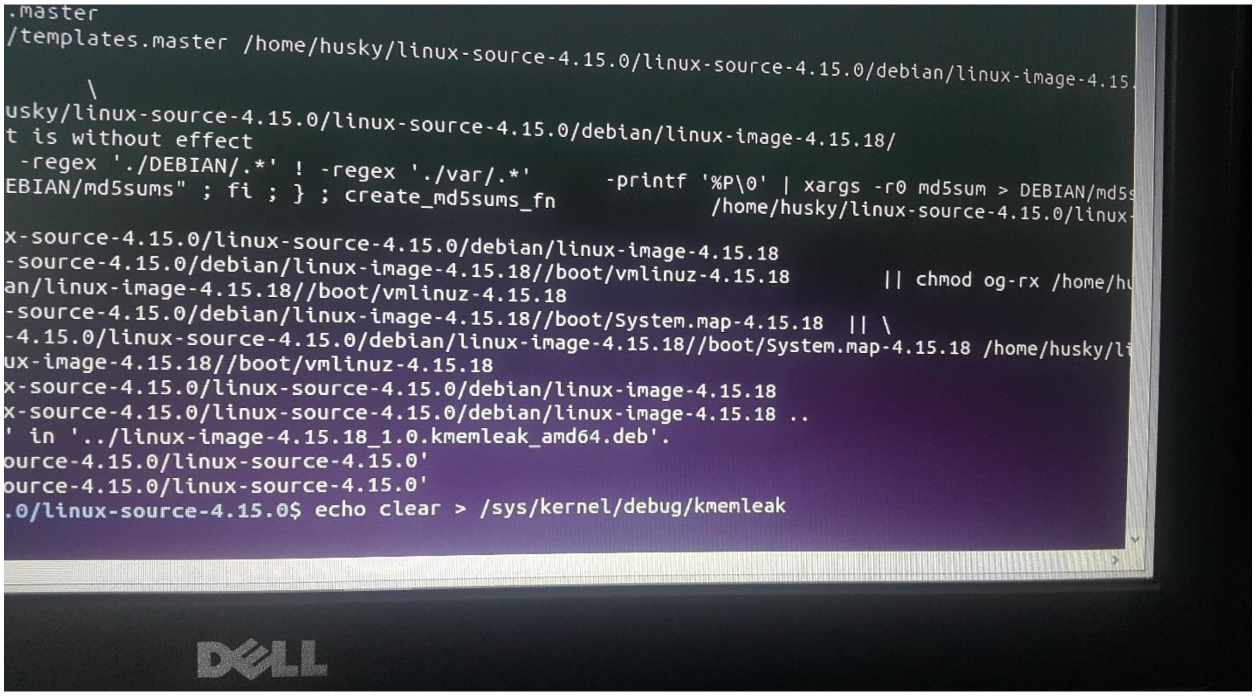
```
.15.0/linux-source-4.15.0/debian/linux-image-4.15.18
15.0/debian/linux-image-4.15.18//boot/vmlinuz-4.15.18        || chmod og-rx /home/hu
nage-4.15.18//boot/vmlinuz-4.15.18
15.0/debian/linux-image-4.15.18//boot/System.map-4.15.18  || \
nux-source-4.15.0/debian/linux-image-4.15.18//boot/System.map-4.15.18 /home/husky/li
.15.18//boot/vmlinuz-4.15.18
.15.0/linux-source-4.15.0/debian/linux-image-4.15.18
.15.0/linux-source-4.15.0/debian/linux-image-4.15.18 ..
inux-image-4.15.18_1.0.kmemleak_amd64.deb'.
.0/linux-source-4.15.0'
.0/linux-source-4.15.0'
source-4.15.0$ sudo apt install libssl-dev
```

Figure 20: Installing SSL library



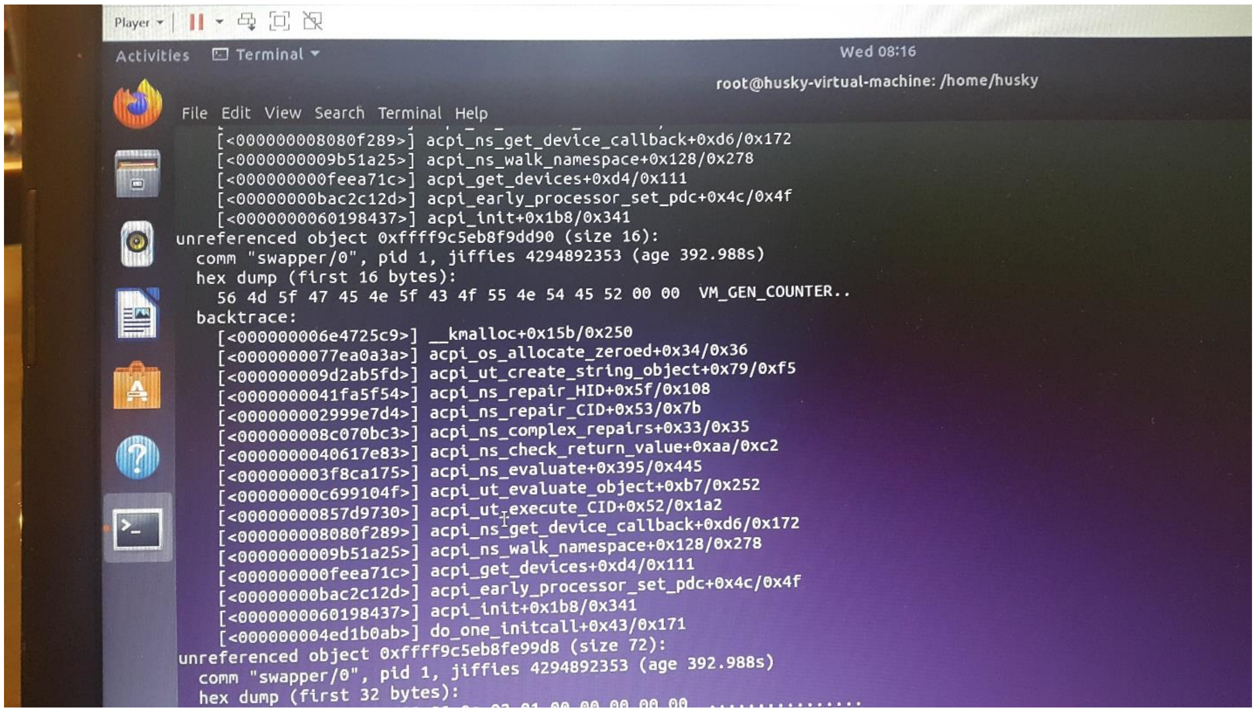
```
husky@husky-virtual-machine:~$ cd linux-source-4.15.0/
husky@husky-virtual-machine:~/linux-source-4.15.0$ ls
debian          linux-image-4.15.18_1.0.kmemleak_amd64.deb
debian.master   linux-source-4.15.0
husky@husky-virtual-machine:~/linux-source-4.15.0$ sudo dpkg -i linux-image-4.15.18_1.0.kmemleak_amd64.deb
```

Figure 21: Install Linux kernel and kmemleak kernel image



```
.master
/templates.master /home/husky/linux-source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15
       \
usky/linux-source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.18/
t is without effect
 -regex './DEBIAN/.*' ! -regex './var/.*'      -printf '%P\0' | xargs -r0 md5sum > DEBIAN/md5s
EBIAN/md5sums" ; fi ; } ; create_md5sums_fn                /home/husky/linux-source-4.15.0/linux

x-source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.18
-source-4.15.0/debian/linux-image-4.15.18//boot/vmlinuz-4.15.18        || chmod og-rx /home/hu
an/linux-image-4.15.18//boot/vmlinuz-4.15.18
-source-4.15.0/debian/linux-image-4.15.18//boot/System.map-4.15.18  || \
-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.18//boot/System.map-4.15.18 /home/husky/li
ux-image-4.15.18//boot/vmlinuz-4.15.18
x-source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.18
x-source-4.15.0/linux-source-4.15.0/debian/linux-image-4.15.18 ..
' in '../linux-image-4.15.18_1.0.kmemleak_amd64.deb'.
ource-4.15.0/linux-source-4.15.0'
ource-4.15.0/linux-source-4.15.0'
.0/linux-source-4.15.0$ echo clear > /sys/kernel/debug/kmemleak
```

Figure 22: Run kmemleak to view if there is any memory leak


Player
Activities
Terminal
Wed 08:16
root@husky-virtual-machine: /home/husky
File Edit View Search Terminal Help

```
    [<000000008080f289>] acpi_ns_get_device_callback+0xd6/0x172
    [<000000009b51a25>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000feea71c>] acpi_get_devices+0xd4/0x111
    [<00000000bac2c12d>] acpi_early_processor_set_pdc+0x4c/0x4f
    [<0000000060198437>] acpi_init+0x1b8/0x341
unreferenced object 0xffff9c5eb8f9dd90 (size 16):
  comm "swapper/0", pid 1, jiffies 4294892353 (age 392.988s)
  hex dump (first 16 bytes):
    56 4d 5f 47 45 4e 5f 43 4f 55 4e 54 45 52 00 00  VM_GEN_COUNTER..
  backtrace:
    [<000000006e4725c9>] __kmalloc+0x15b/0x250
    [<0000000077ea0a3a>] acpi_os_allocate_zeroed+0x34/0x36
    [<000000009d2ab5fd>] acpi_ut_create_string_object+0x79/0xf5
    [<0000000041fa5f54>] acpi_ns_repair_HID+0x5f/0x108
    [<000000002999e7d4>] acpi_ns_repair_CID+0x53/0x7b
    [<000000008c070bc3>] acpi_ns_complex_repairs+0x33/0x35
    [<0000000040617e83>] acpi_ns_check_return_value+0xaa/0xc2
    [<000000003f8ca175>] acpi_ns_evaluate+0x395/0x445
    [<00000000c699104f>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000857d9730>] acpi_ut_execute_CID+0x52/0x1a2
    [<000000008080f289>] acpi_ns_get_device_callback+0xd6/0x172
    [<000000009b51a25>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000feea71c>] acpi_get_devices+0xd4/0x111
    [<00000000bac2c12d>] acpi_early_processor_set_pdc+0x4c/0x4f
    [<0000000060198437>] acpi_init+0x1b8/0x341
    [<000000004ed1b0ab>] do_one_initcall+0x43/0x171
unreferenced object 0xffff9c5eb8fe99d8 (size 72):
  comm "swapper/0", pid 1, jiffies 4294892353 (age 392.988s)
  hex dump (first 32 bytes):
```

Figure 23: Kmemleak output showing memory leakage

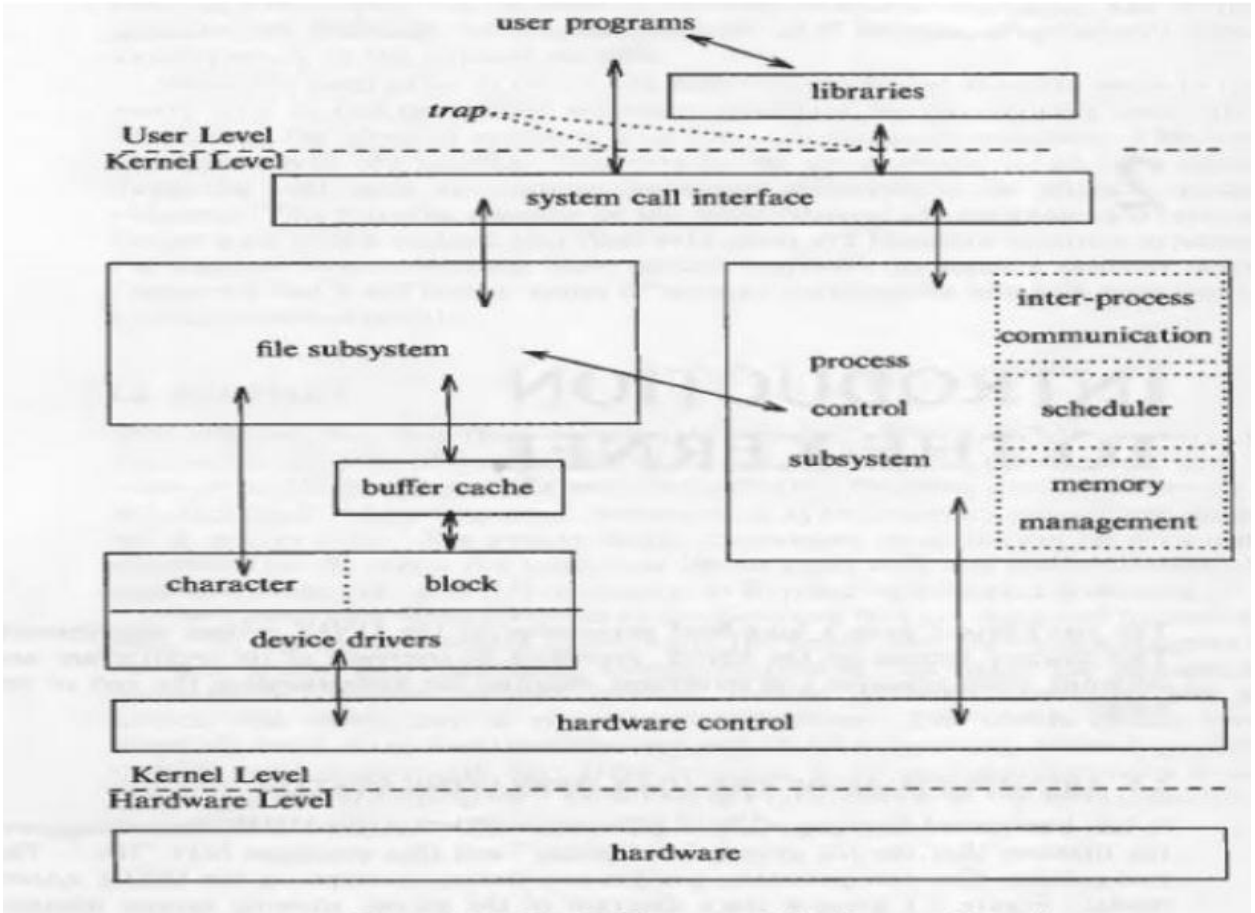


Figure 2.1. Block Diagram of the System Kernel

Figure 24: Reena Murali. (2013). Block Diagram of the System Kernel [Figure]. Os concepts. Retrieved from https://www.slideshare.net/SUDHEESHPENATHU/os-concepts-28589673

Appendix B: More Information

Table 3: The remaining thirteen new suspected memory leak

```
unreferenced object 0xffff942ab9939948 (size 72):
  comm "swapper/0", pid 1, jiffies 4294892400 (age 1173.244s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 0e 02 01 00 02 00 00 01  ................
    7e f1 02 40 54 b6 ff ff 08 00 00 00 00 00 00 00  ~..@T...........
  backtrace:
    [<000000002517a863>] kmem_cache_alloc+0xf8/0x1e0
    [<000000002b330b88>] acpi_ut_allocate_object_desc_dbg+0x62/0x10c
    [<00000000f5fb5003>] acpi_ut_create_internal_object_dbg+0x53/0x117
    [<0000000025f2a706>] acpi_ds_build_internal_object+0xe5/0x1cc
    [<0000000019bb4fb0>] acpi_ds_build_internal_package_obj+0x1ea/0x331
    [<00000000f66c35a7>] acpi_ds_eval_data_object_operands+0x17d/0x219
    [<000000005c7c7ddb>] acpi_ds_exec_end_op+0x4b5/0x764
    [<000000007b93b1b9>] acpi_ps_parse_loop+0xa38/0xadc
    [<000000009fbe3906>] acpi_ps_parse_aml+0x1ac/0x4bd
    [<000000009f450676>] acpi_ds_execute_arguments+0x18b/0x1d2
    [<00000000033ee6ee>] acpi_ds_get_package_arguments+0xfd/0x12c
    [<00000000badb67db>] acpi_ns_init_one_object+0xeb/0x155
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000edaf2427>] acpi_walk_namespace+0xf9/0x144
    [<00000000c7ffe1b1>] acpi_ns_initialize_objects+0x108/0x1f5
    [<00000000fd360aaf>] acpi_initialize_objects+0x4b/0xd5
```

```
unreferenced object 0xffff942ab9939e10 (size 72):
  comm "swapper/0", pid 1, jiffies 4294892424 (age 1173.148s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 0e 02 01 00 00 00 00 00  ................
    c0 17 d4 b8 2a 94 ff ff 0e 00 00 00 00 00 00 00  ....*...........
  backtrace:
    [<000000002517a863>] kmem_cache_alloc+0xf8/0x1e0
    [<000000002b330b88>] acpi_ut_allocate_object_desc_dbg+0x62/0x10c
    [<00000000f5fb5003>] acpi_ut_create_internal_object_dbg+0x53/0x117
    [<00000000cfe8a0c3>] acpi_ut_create_string_object+0x4a/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<00000000df8728c3>] acpi_early_processor_set_pdc+0x4c/0x4f
    [<00000000031ef9e1>] acpi_init+0x1b8/0x341
```

```
unreferenced object 0xffff942ab8d417c0 (size 16):
  comm "swapper/0", pid 1, jiffies 4294892424 (age 1173.152s)
  hex dump (first 16 bytes):
    56 4d 5f 47 45 4e 5f 43 4f 55 4e 54 45 52 00 00  VM_GEN_COUNTER..
  backtrace:
    [<0000000007a58044>] __kmalloc+0x15b/0x250
    [<00000000b1765366>] acpi_os_allocate_zeroed+0x34/0x36
    [<0000000049f3fede>] acpi_ut_create_string_object+0x79/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<00000000df8728c3>] acpi_early_processor_set_pdc+0x4c/0x4f
    [<0000000031ef9e1>] acpi_init+0x1b8/0x341
    [<00000000a68a04cd>] do_one_initcall+0x43/0x171
```

```
unreferenced object 0xffff942ab9939b40 (size 72):
  comm "swapper/0", pid 1, jiffies 4294892424 (age 1173.152s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 0e 02 01 00 00 00 00 00  ................
    80 10 d4 b8 2a 94 ff ff 08 00 00 00 00 00 00 00  ....*...........
  backtrace:
    [<000000002517a863>] kmem_cache_alloc+0xf8/0x1e0
    [<000000002b330b88>] acpi_ut_allocate_object_desc_dbg+0x62/0x10c
    [<00000000f5fb5003>] acpi_ut_create_internal_object_dbg+0x53/0x117
    [<00000000cfe8a0c3>] acpi_ut_create_string_object+0x4a/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<00000000df8728c3>] acpi_early_processor_set_pdc+0x4c/0x4f
    [<00000000031ef9e1>] acpi_init+0x1b8/0x341
```

```
unreferenced object 0xffff942ab8d41080 (size 16):
  comm "swapper/0", pid 1, jiffies 4294892424 (age 1173.152s)
  hex dump (first 16 bytes):
    41 43 50 49 30 30 31 32 00 00 00 00 00 00 00 00  ACPI0012........
  backtrace:
    [<0000000007a58044>] __kmalloc+0x15b/0x250
    [<00000000b1765366>] acpi_os_allocate_zeroed+0x34/0x36
    [<0000000049f3fede>] acpi_ut_create_string_object+0x79/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<00000000df8728c3>] acpi_early_processor_set_pdc+0x4c/0x4f
    [<0000000031ef9e1>] acpi_init+0x1b8/0x341
    [<00000000a68a04cd>] do_one_initcall+0x43/0x171
```

```
unreferenced object 0xffff942ab9939630 (size 72):
  comm "swapper/0", pid 1, jiffies 4294892425 (age 1173.156s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 0e 02 01 00 00 00 00 00  ................
    e0 1e d4 b8 2a 94 ff ff 0e 00 00 00 00 00 00 00  ....*...........
  backtrace:
    [<000000002517a863>] kmem_cache_alloc+0xf8/0x1e0
    [<000000002b330b88>] acpi_ut_allocate_object_desc_dbg+0x62/0x10c
    [<00000000f5fb5003>] acpi_ut_create_internal_object_dbg+0x53/0x117
    [<00000000cfe8a0c3>] acpi_ut_create_string_object+0x4a/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2

    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172

    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278

    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111

    [<000000008b6803ff>] acpi_ec_dsdt_probe+0x43/0x78

    [<00000000af939fd7>] acpi_init+0x1bd/0x341
```

```
unreferenced object 0xffff942ab8d41ee0 (size 16):

  comm "swapper/0", pid 1, jiffies 4294892425 (age 1173.156s)

  hex dump (first 16 bytes):

    56 4d 5f 47 45 4e 5f 43 4f 55 4e 54 45 52 00 00  VM_GEN_COUNTER..

  backtrace:

    [<000000007a58044>] __kmalloc+0x15b/0x250

    [<00000000b1765366>] acpi_os_allocate_zeroed+0x34/0x36

    [<0000000049f3fede>] acpi_ut_create_string_object+0x79/0xf5

    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108

    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b

    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35

    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2

    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445

    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252

    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<000000008b6803ff>] acpi_ec_dsdt_probe+0x43/0x78
    [<00000000af939fd7>] acpi_init+0x1bd/0x341
    [<00000000a68a04cd>] do_one_initcall+0x43/0x171
```

```
unreferenced object 0xffff942ab99399d8 (size 72):
  comm "swapper/0", pid 1, jiffies 4294892425 (age 1173.156s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 0e 02 01 00 00 00 00 00  ................
    60 14 d4 b8 2a 94 ff ff 08 00 00 00 00 00 00 00  `...*...........
  backtrace:
    [<000000002517a863>] kmem_cache_alloc+0xf8/0x1e0
    [<000000002b330b88>] acpi_ut_allocate_object_desc_dbg+0x62/0x10c
    [<00000000f5fb5003>] acpi_ut_create_internal_object_dbg+0x53/0x117
    [<00000000cfe8a0c3>] acpi_ut_create_string_object+0x4a/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<000000008b6803ff>] acpi_ec_dsdt_probe+0x43/0x78
    [<00000000af939fd7>] acpi_init+0x1bd/0x341
```

```
unreferenced object 0xffff942ab8d41460 (size 16):
  comm "swapper/0", pid 1, jiffies 4294892425 (age 1173.164s)
  hex dump (first 16 bytes):
    41 43 50 49 30 30 31 32 00 00 00 00 00 00 00 00  ACPI0012........
  backtrace:
    [<0000000007a58044>] __kmalloc+0x15b/0x250
    [<00000000b1765366>] acpi_os_allocate_zeroed+0x34/0x36
    [<0000000049f3fede>] acpi_ut_create_string_object+0x79/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<000000008b6803ff>] acpi_ec_dsdt_probe+0x43/0x78
    [<00000000af939fd7>] acpi_init+0x1bd/0x341
    [<00000000a68a04cd>] do_one_initcall+0x43/0x171
```

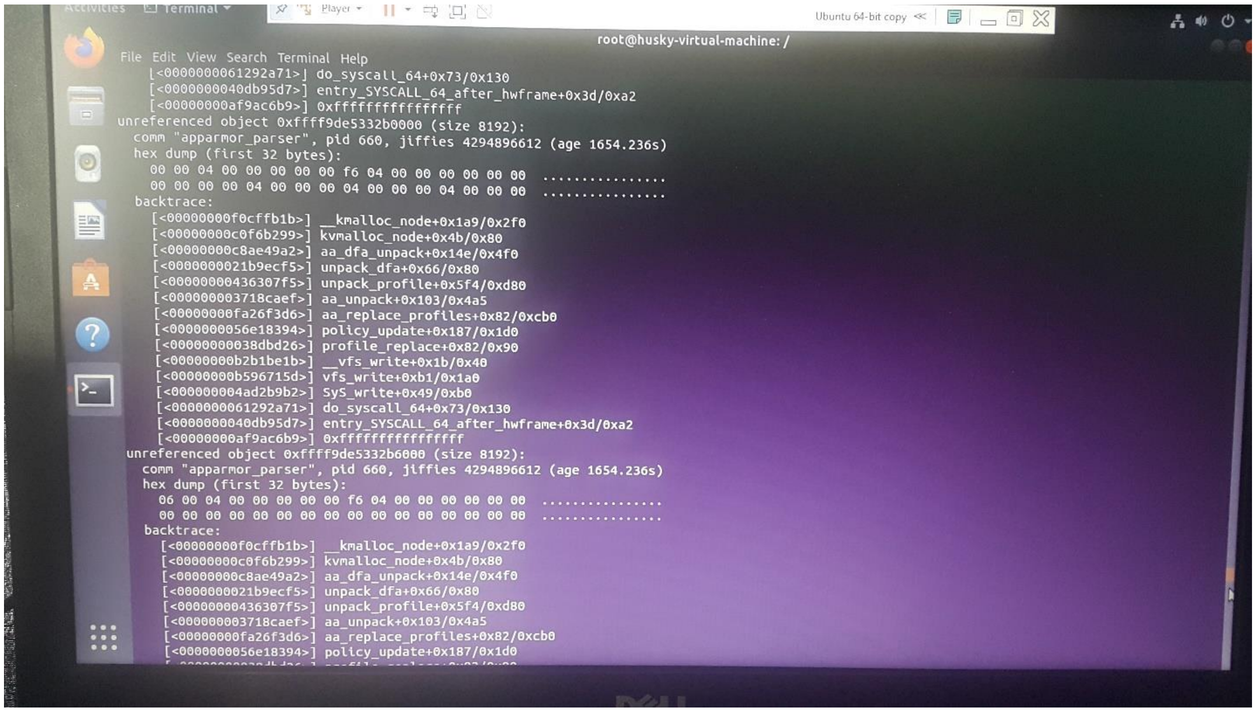
```
unreferenced object 0xffff942ab9939168 (size 72):
  comm "swapper/0", pid 1, jiffies 4294892425 (age 1173.164s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 0e 02 01 00 00 00 00 00  ................
    50 1f d4 b8 2a 94 ff ff 0e 00 00 00 00 00 00 00  P...*...........
  backtrace:
    [<000000002517a863>] kmem_cache_alloc+0xf8/0x1e0
    [<000000002b330b88>] acpi_ut_allocate_object_desc_dbg+0x62/0x10c
    [<00000000f5fb5003>] acpi_ut_create_internal_object_dbg+0x53/0x117
    [<00000000cfe8a0c3>] acpi_ut_create_string_object+0x4a/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<000000006ff3d6c5>] acpi_processor_init+0x40/0x62
    [<00000000b27b33c0>] acpi_scan_init+0x1f/0x22c
```

```
unreferenced object 0xffff942ab8d41f50 (size 16):
  comm "swapper/0", pid 1, jiffies 4294892425 (age 1173.164s)
  hex dump (first 16 bytes):
    56 4d 5f 47 45 4e 5f 43 4f 55 4e 54 45 52 00 00  VM_GEN_COUNTER..
  backtrace:
    [<000000007a58044>] __kmalloc+0x15b/0x250
    [<00000000b1765366>] acpi_os_allocate_zeroed+0x34/0x36
    [<0000000049f3fede>] acpi_ut_create_string_object+0x79/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<000000006ff3d6c5>] acpi_processor_init+0x40/0x62
    [<00000000b27b33c0>] acpi_scan_init+0x1f/0x22c
    [<00000000a733c9a6>] acpi_init+0x2f2/0x341
```

```
unreferenced object 0xffff942ab99390d8 (size 72):
  comm "swapper/0", pid 1, jiffies 4294892425 (age 1173.172s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 0e 02 01 00 00 00 00 00  ................
    c0 1e d4 b8 2a 94 ff ff 08 00 00 00 00 00 00 00  ....*...........
  backtrace:
    [<000000002517a863>] kmem_cache_alloc+0xf8/0x1e0
    [<000000002b330b88>] acpi_ut_allocate_object_desc_dbg+0x62/0x10c
    [<00000000f5fb5003>] acpi_ut_create_internal_object_dbg+0x53/0x117
    [<00000000cfe8a0c3>] acpi_ut_create_string_object+0x4a/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
    [<00000000ca650011>] acpi_ns_evaluate+0x395/0x445
    [<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252
    [<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2
    [<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172
    [<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278
    [<00000000ceef9850>] acpi_get_devices+0xd4/0x111
    [<000000006ff3d6c5>] acpi_processor_init+0x40/0x62
    [<00000000b27b33c0>] acpi_scan_init+0x1f/0x22c
```

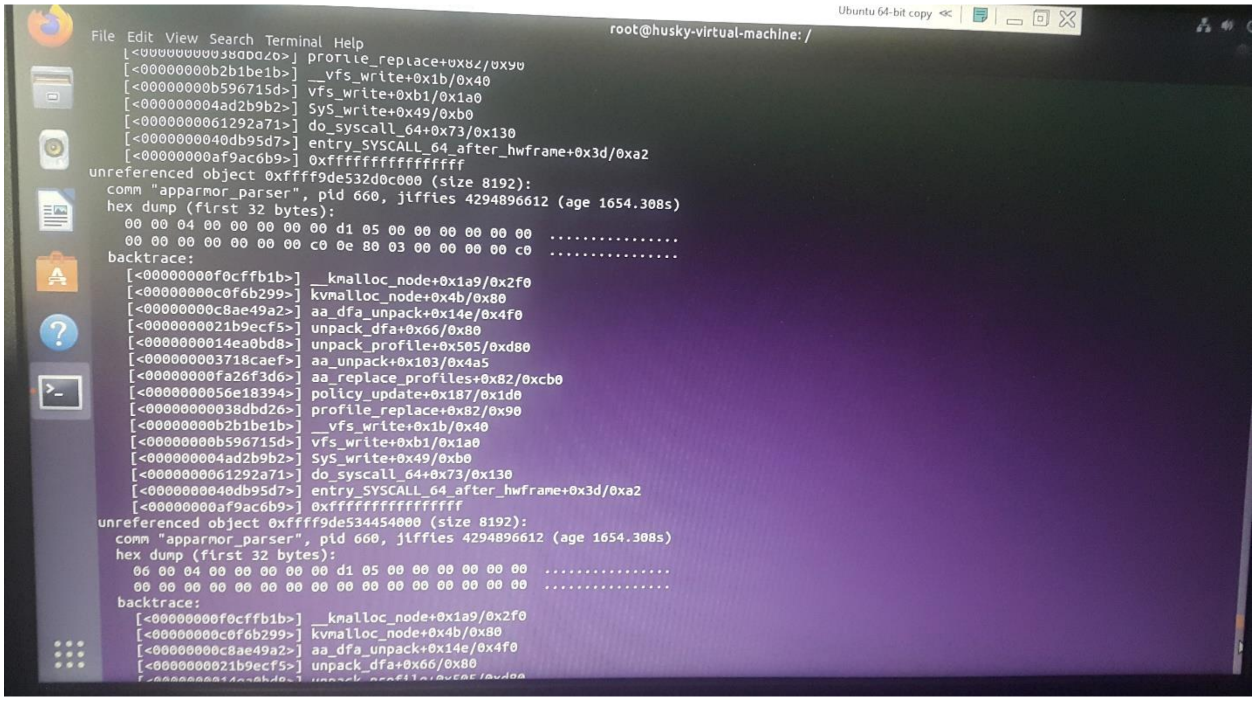
```
unreferenced object 0xffff942ab8d41ec0 (size 16):
  comm "swapper/0", pid 1, jiffies 4294892425 (age 1173.172s)
  hex dump (first 16 bytes):
    41 43 50 49 30 30 31 32 00 00 00 00 00 00 00 00  ACPI0012........
  backtrace:
    [<0000000007a58044>] __kmalloc+0x15b/0x250
    [<00000000b1765366>] acpi_os_allocate_zeroed+0x34/0x36
    [<0000000049f3fede>] acpi_ut_create_string_object+0x79/0xf5
    [<00000000961dcb33>] acpi_ns_repair_HID+0x5f/0x108
    [<00000000b7344f17>] acpi_ns_repair_CID+0x53/0x7b
    [<00000000f823f531>] acpi_ns_complex_repairs+0x33/0x35
    [<00000000ac9086ad>] acpi_ns_check_return_value+0xaa/0xc2
```

```
[<00000000ca650011>] acpi_ns_evaluate+0x395/0x445

[<000000007e7b4d58>] acpi_ut_evaluate_object+0xb7/0x252

[<00000000efe25f55>] acpi_ut_execute_CID+0x52/0x1a2

[<00000000ffc430e6>] acpi_ns_get_device_callback+0xd6/0x172

[<00000000913d2796>] acpi_ns_walk_namespace+0x128/0x278

[<00000000ceef9850>] acpi_get_devices+0xd4/0x111

[<000000006ff3d6c5>] acpi_processor_init+0x40/0x62

[<00000000b27b33c0>] acpi_scan_init+0x1f/0x22c

[<00000000a733c9a6>] acpi_init+0x2f2/0x341
```


Activities
Terminal
Player
Ubuntu 64-bit copy
root@husky-virtual-machine: /
File Edit View Search Terminal Help

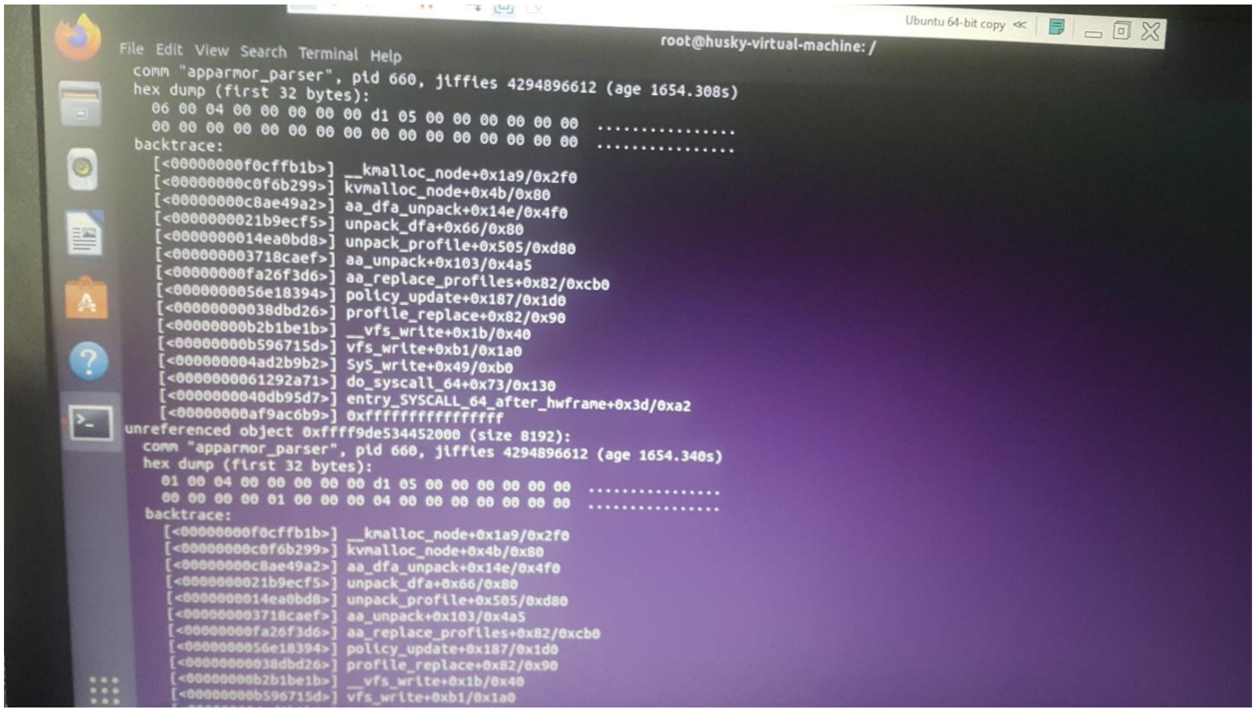
```
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de5332b0000 (size 8192):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.236s)
  hex dump (first 32 bytes):
    00 00 04 00 00 00 00 00 f6 04 00 00 00 00 00 00  ................
    00 00 00 00 04 00 00 00 04 00 00 00 04 00 00 00  ................
  backtrace:
    [<00000000f0cffb1b>] __kmalloc_node+0x1a9/0x2f0
    [<00000000c0f6b299>] kvmalloc_node+0x4b/0x80
    [<00000000c8ae49a2>] aa_dfa_unpack+0x14e/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<00000000436307f5>] unpack_profile+0x5f4/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de5332b6000 (size 8192):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.236s)
  hex dump (first 32 bytes):
    06 00 04 00 00 00 00 00 f6 04 00 00 00 00 00 00  ................
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  backtrace:
    [<00000000f0cffb1b>] __kmalloc_node+0x1a9/0x2f0
    [<00000000c0f6b299>] kvmalloc_node+0x4b/0x80
    [<00000000c8ae49a2>] aa_dfa_unpack+0x14e/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<00000000436307f5>] unpack_profile+0x5f4/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
```

Figure 25: Kmemleak showing empty data leak for the 73 hours



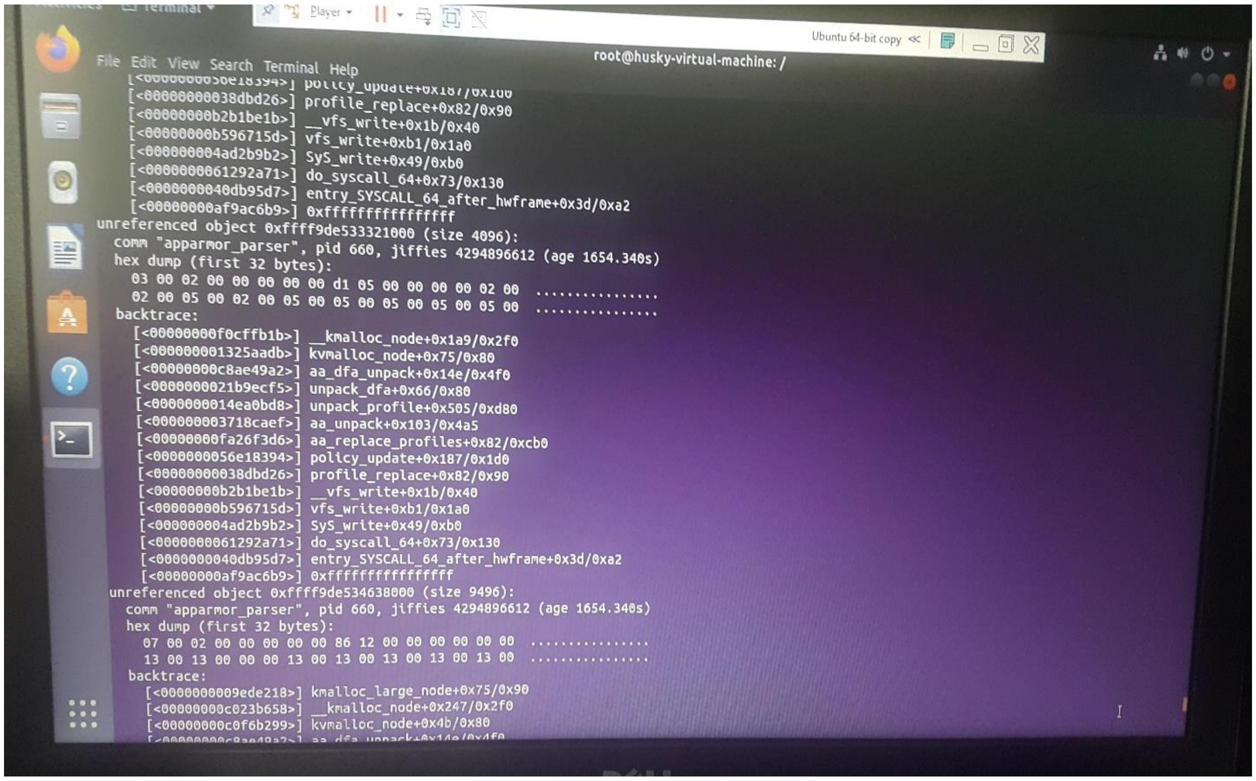
```
File  Edit  View  Search  Terminal  Help
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de532d0c000 (size 8192):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.308s)
  hex dump (first 32 bytes):
    00 00 04 00 00 00 00 00 d1 05 00 00 00 00 00 00  ................
    00 00 00 00 00 00 00 c0 0e 80 03 00 00 00 00 c0  ................
  backtrace:
    [<00000000f0cffb1b>] __kmalloc_node+0x1a9/0x2f0
    [<00000000c0f6b299>] kvmalloc_node+0x4b/0x80
    [<00000000c8ae49a2>] aa_dfa_unpack+0x14e/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<0000000014ea0bd8>] unpack_profile+0x505/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de534454000 (size 8192):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.308s)
  hex dump (first 32 bytes):
    06 00 04 00 00 00 00 00 d1 05 00 00 00 00 00 00  ................
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  backtrace:
    [<00000000f0cffb1b>] __kmalloc_node+0x1a9/0x2f0
    [<00000000c0f6b299>] kvmalloc_node+0x4b/0x80
    [<00000000c8ae49a2>] aa_dfa_unpack+0x14e/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<0000000014ea0bd8>] unpack_profile+0x505/0xd80
```

Figure 26: Example of kmemleak showing empty data leak for the 73 hours



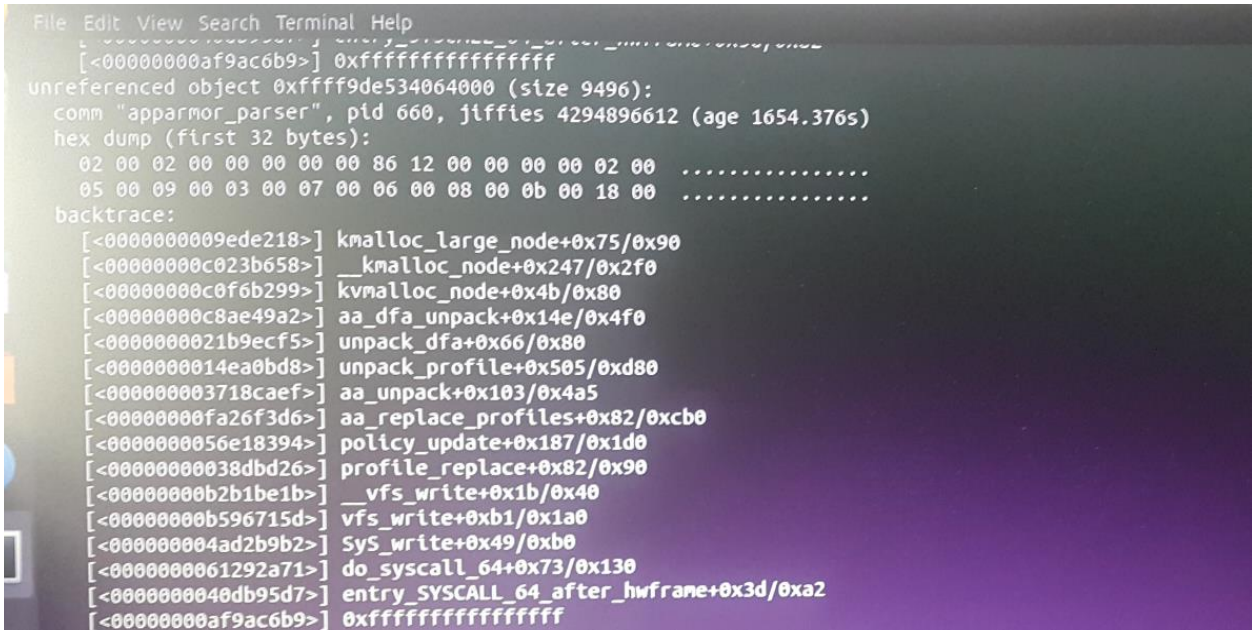
```
File  Edit  View  Search  Terminal  Help
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.308s)
  hex dump (first 32 bytes):
    06 00 04 00 00 00 00 00 d1 05 00 00 00 00 00 00  ................
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  backtrace:
    [<00000000f0cffb1b>] __kmalloc_node+0x1a9/0x2f0
    [<00000000c0f6b299>] kvmalloc_node+0x4b/0x80
    [<00000000c8ae49a2>] aa_dfa_unpack+0x14e/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<0000000014ea0bd8>] unpack_profile+0x505/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de534452000 (size 8192):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.340s)
  hex dump (first 32 bytes):
    01 00 04 00 00 00 00 00 d1 05 00 00 00 00 00 00  ................
    00 00 00 00 01 00 00 00 04 00 00 00 00 00 00 00  ................
  backtrace:
    [<00000000f0cffb1b>] __kmalloc_node+0x1a9/0x2f0
    [<00000000c0f6b299>] kvmalloc_node+0x4b/0x80
    [<00000000c8ae49a2>] aa_dfa_unpack+0x14e/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<0000000014ea0bd8>] unpack_profile+0x505/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
```

Figure 27: Next example of kmemleak showing empty data leak for the 73 hours


Player
Ubuntu 64-bit copy
root@husky-virtual-machine: /
File Edit View Search Terminal Help

```
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de533321000 (size 4096):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.340s)
  hex dump (first 32 bytes):
    03 00 02 00 00 00 00 00 d1 05 00 00 00 00 02 00  ................
    02 00 05 00 02 00 05 00 05 00 05 00 05 00 05 00  ................
  backtrace:
    [<00000000f0cffb1b>] __kmalloc_node+0x1a9/0x2f0
    [<000000001325aadb>] kvmalloc_node+0x75/0x80
    [<00000000c8ae49a2>] aa_dfa_unpack+0x14e/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<0000000014ea0bd8>] unpack_profile+0x505/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de534638000 (size 9496):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.340s)
  hex dump (first 32 bytes):
    07 00 02 00 00 00 00 00 86 12 00 00 00 00 00 00  ................
    13 00 13 00 00 00 13 00 13 00 13 00 13 00 13 00  ................
  backtrace:
    [<000000009ede218>] kmalloc_large_node+0x75/0x90
    [<00000000c023b658>] __kmalloc_node+0x247/0x2f0
    [<00000000c0f6b299>] kvmalloc_node+0x4b/0x80
```

Figure 28: Another example of kmemleak showing empty data leak for the 73 hours



```
    [<00000000af9ac6b9>] 0xffffffffffffffff
unreferenced object 0xffff9de534064000 (size 9496):
  comm "apparmor_parser", pid 660, jiffies 4294896612 (age 1654.376s)
  hex dump (first 32 bytes):
    02 00 02 00 00 00 00 00 86 12 00 00 00 00 02 00  ................
    05 00 09 00 03 00 07 00 06 00 08 00 0b 00 18 00  ................
  backtrace:
    [<000000009ede218>] kmalloc_large_node+0x75/0x90
    [<00000000c023b658>] __kmalloc_node+0x247/0x2f0
    [<00000000c0f6b299>] kvmalloc_node+0x4b/0x80
    [<00000000c8ae49a2>] aa_dfa_unpack+0x14e/0x4f0
    [<0000000021b9ecf5>] unpack_dfa+0x66/0x80
    [<0000000014ea0bd8>] unpack_profile+0x505/0xd80
    [<000000003718caef>] aa_unpack+0x103/0x4a5
    [<00000000fa26f3d6>] aa_replace_profiles+0x82/0xcb0
    [<0000000056e18394>] policy_update+0x187/0x1d0
    [<0000000038dbd26>] profile_replace+0x82/0x90
    [<00000000b2b1be1b>] __vfs_write+0x1b/0x40
    [<00000000b596715d>] vfs_write+0xb1/0x1a0
    [<000000004ad2b9b2>] SyS_write+0x49/0xb0
    [<0000000061292a71>] do_syscall_64+0x73/0x130
    [<0000000040db95d7>] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
    [<00000000af9ac6b9>] 0xffffffffffffffff
```

Figure 29: Another kmemleak showing empty data leak for the 73 hours